CHALMERS

Web Services Integration Method

Master of Science Thesis in the Programme Networks and Distributed sytstems

CHANGBIN WANG
YUAN XU

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, October 2010

Web services integration method

Changbin Wang
Cooperation with Yuan Xu

Examiner: Sven-Arne Andréasson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2010

Abstract

Cloud computing as next generation computing paradigm has been accepted widely with growing-up of Internet technology and success of World Wide Web, which has the potential to transform a large part of IT industry, making software more attractive and even change the way IT product is designed and purchased. Cloud computing is all about services of various levels based upon Internet or Intranet including Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-service (IaaS) and so on. The most wonderful part of what's happening around the web and cloud computing is being able to expose data in a very pervasive, scalable and uniform way. Enterprises around the world can expose their data through web services loose-coupling and efficiently, in which way we can build more powerful and interesting applications once data is liberated from different organizations. WS-* and REpresentational State Transfer (REST) as two dominant integration methodologies are supporting widespread cloud services.

This paper introduces a study with objective of evaluating new integration technology REpresentational State Transfer (REST) and how to design and development RESTful web services in the cloud environment based on evaluation from software architecture perspective. In addition to, the paper presents security solution for RESTful Web services as well as some critical decisions and main concerns for implementation of the successful and real RESTful web services. Meanwhile, the paper reports on literature study about cloud computing and theoretical comparison between two integration methodologies – REpresentational State Transfer and WS-*, and so on.

Keywords: REpresentational State Transfer (REST), Integration method, OAuth, cloud computing, WS-*, Software architecture, Software architectural style, web services.

Acknowledgment

I would like to express my sincere gratitude to my examiner Sven-Arne Andréasson and supervisor Mr. Jesper Forslund in Logica for the help, support and guidance during my master thesis study.

I also would like to say thanks to my partner Xu, Yuan, who gives me lots of good advice and support.

Contents

1. Introduction

Gartner said cloud computing will be as influential as E-business and the confusion and contradiction that surround the term cloud computing also indicts the potential to changing the status quo in the IT market [1]. Market-research firm IDC expects IT cloud-services spending will grow from about $16 billion in 2008 to about $42 billion by 2012. IDC (2008) also predicts cloud-computing spending will account for 25 percent of annual IT expenditure growth by 2012 and nearly a third of the growth the following year [2]. Even up to now, there is no consensus in the cloud computing definition. Cloud Computing has been talked about, blogged about, written about and been featured in the title of workshops, conferences, and even magazines. Nevertheless, confusion remains about exactly what it is and when it's useful, causing Oracle's CEO to vent his frustration:

"The interesting thing about Cloud Computing is that we've redefined Cloud Computing to include everything that we already do.... I don't understand what we would do differently in the light of Cloud Computing other than change the wording of some of our ads."

Larry Ellison, quoted in the *Wall Street Journal*, September 26, 2008

These remarks are echoed more mildly by Hewlett-Packard's Vice President of European Software Sales:

"A lot of people are jumping on the [cloud] bandwagon, but I have not heard two people say the same thing about it. There are multiple definitions out there of the cloud."

Andy Isherwood, quoted in *ZDnet News,* December 11, 2008

Although there are still some different sounds, criticisms and natural obstacles [3] about cloud computing many IT industry leaders such as Google, IBM, Microsoft and Amazon have landed in this new continent to catch the trend. We can say, in many ways, cloud computing is simply a metaphor for the internet and the increasing movement of compute and data resources onto the Web [4].

New web development and deployment platforms are arising based on Cloud computing and Software-as-a-Service. These new Internet-enabled platforms had appeared enabling open collaboration and creation. These platforms introduce a new method of delivering software applications. Customers access the application over the Internet using industry-standard browsers or Web Services clients [5]. Online software delivery is now conceived and defined as Software-as-Service (SaaS). It is growing into a mainstream option for software-based solutions and this will impact most of the enterprise IT departments in the next three years [6]. Chou declares that SaaS is the next step in the software industry, because it fundamentally alters the economics of software [7].
"Something momentous is happening in the software business. Bill Gates of Microsoft calls it "the next sea change". Analysts call it a "tectonic shift" in the industry. Trade publications hail it as "the next big thing"."

The Economist journal, April 20, 2006

This market will be growing in the next years, according to Gartner; trends in SaaS business are: By 2011, 25% of new business software will be delivered as SaaS. By 2012, business process management suites (BPMSs) will be embedded in at least 40% of all new SaaS offerings. By 2012, more than 66% of independent software vendors (ISVs) will offer some of their services as SaaS [8]. Also, IDC estimates customers spending on SaaS solutions to increase to $14.8 billion by 2011. After focusing on these crucial business aspects and previous prediction, we can aware that the importance and quantity about software that will be delivered with SaaS in Cloud computing.

Roy Thomas Fielding first introduces REpresentational State Transfer (REST) architectural style in his PhD dissertation [9]. REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations. [10] The first edition of REST was developed between October 1994 and August 1995, primarily as a means for communicating Web concepts while developing the HTTP/1.0 specification and the initial HTTP/1.1 proposal. We can say REST's all built around the web, thanks to web's success, and the web is all around us. So REST as integration methodology gives us an alternative to construct SOA. According the statistical figures of Programmable web [11] in May 2010 REST as dominator integration protocol has had 72% market share.

2. Project description

The research reported in this paper was conducted between March 2010 and June 2010. The research depends on the project Pomodoro which is a collaborative study proposed by Logica cooperating with Chalmers University and Web2Aid. The implementing of Pomodoro technology is chosen for evaluating new technologies. Cloud Computing Service Provider is a sub-project that supports different services through open APIs under cloud computing environment. Service Provider is the data owner of the Pomodoro consumers including iPhone mobile application, Android mobile application and mainstream web frameworks and its job is to publish an open API that others can use to access and work with that data.
This project will develop the same product on different platforms and integrate with a central service provider. As an end user you can switch between working with this application on your computer to your phone in no time, without any data loss. The structure of this project is descripted by the following illustration.

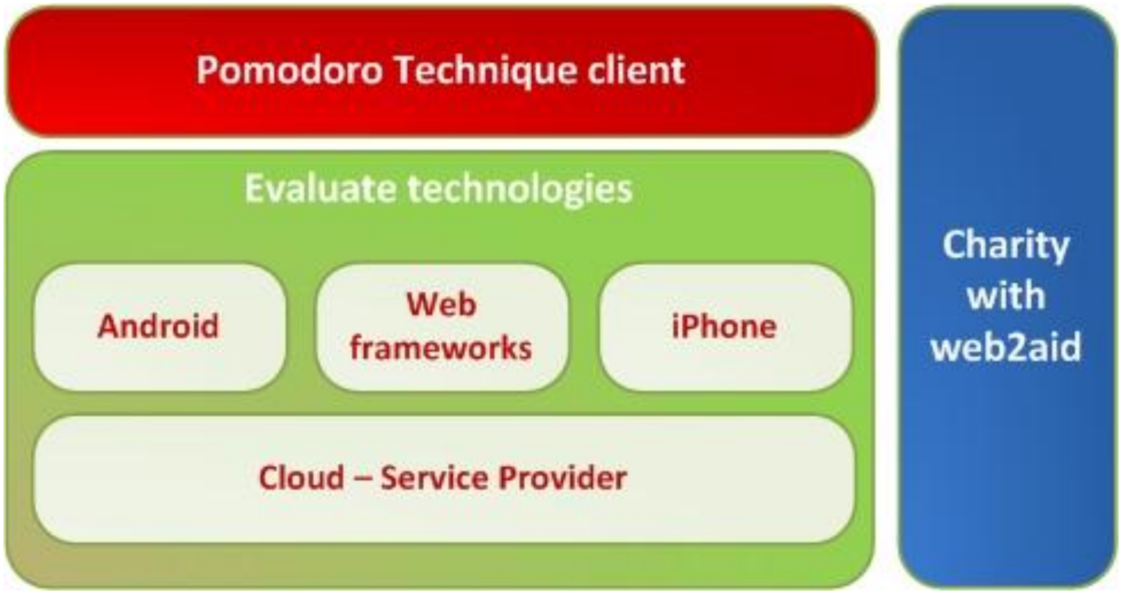


Figure2.1 overview of Pomodoro project

Logica is a business and technology service company, employing 39,000 people across 36 countries. It delivers business consulting, systems integration and outsourcing across all industries and business functions. It creates value by successfully integrating people, business and technology to create effective, sustainable business ecosystems. Their people apply insight to create innovative answers to your business needs.

3. Cloud computing

Cloud computing may be the hottest word in today's IT field. "It's become the phrase du jour", says Gartner senior analyst Ben Pring. Everyone is talking about it, and almost all major IT industrial leading companies said they will focus on it in the next 5 years. In fact cloud computing conception is more about economy rather than technological revolution, it not only is the evolution of the IT technologies but also will change the thinking mode when we talk about the computer, the Internet and the way we use them. So what is "cloud computing"? The "cloud" as the metaphor of Internet exists a long time, but when it combined with "computing" the meaning gets bigger and fuzzier. The NIST published the draft of the NIST working definition of cloud computing give the definition of cloud computing [12]:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The definition may be still confusing, but it gives the most important characteristics of it.

1. On-demand self-services: customers can provision computing capabilities, such as storage, server time, as needed automatically without requiring human interaction with each service's provider.
2. Broad Network access: capabilities are available over the network and accessed through standard mechanisms that promotes use by heterogeneous thin or thick client platform, such as mobile phones, laptops, and PDAs
3. Resource pooling: The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
4. Rapid elasticity: Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumers, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.
5. Measured Service: Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service

When deploy the cloud there are four models exist:

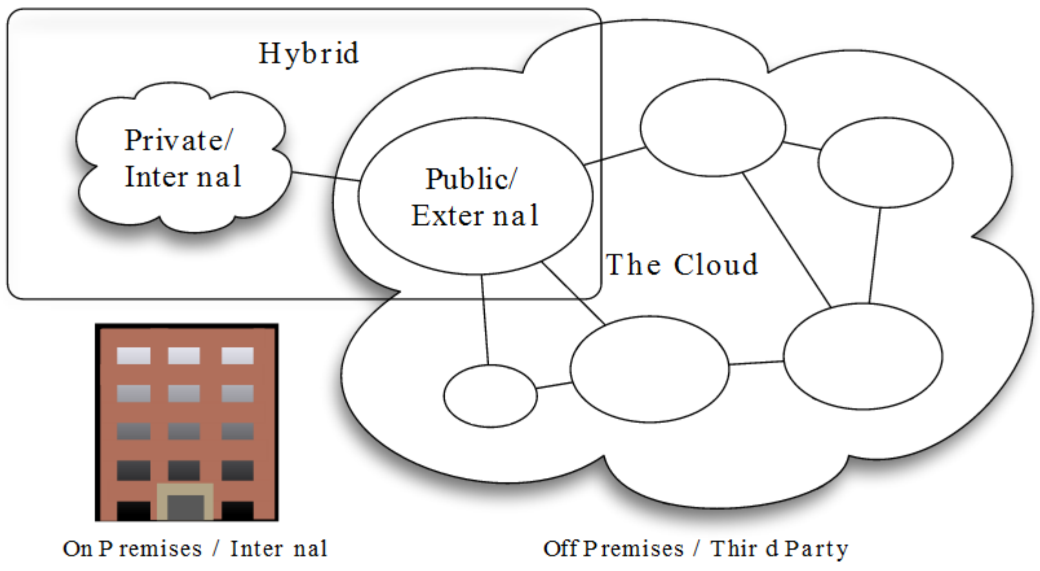


Figure3.1 Cloud computing models

1. Private cloud

   This cloud infrastructure is owned by the sole organization and it can be managed by the organization or a third party but only the organization members can access the services.

2. Community cloud

   This cloud infrastructure can be shared by several related organizations. It also can be managed by these organizations or a third party and may exist on premise or off premise.

3. Public cloud

   This cloud infrastructure is made available to the public to access the services provided by the cloud, the cloud may be owned by anyone who selling the services.

4. Hybrid cloud

   The cloud infrastructure is a composition of two or more clouds which may be included public cloud and private cloud. All the clouds in the infrastructure are bounded together to provide the service to general public and the organization internal use.

When people embrace the “cloud computing”, their concerns are all about the cloud will provide which benefits to them and what kind of services can be provided by it. Essentially there are 10 service models:

1. Cloud Software as a Service(SaaS)

   The capability provided to consumers is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through thin clients’ interface such as a web browser (e.g., web-based email). Consumers do not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

2. Cloud Platform as a Service(PaaS)

   The capability provided to consumers is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. Consumers do not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

3. Cloud Infrastructure as a Service(IaaS)

   The capability provided to consumers is to provision processing, storage, networks, and other fundamental computing resources where consumers are able to deploy and run arbitrary software, which can include operating systems and applications. Consumers not only manage or control the underlying cloud infrastructure but also control over operating systems; storage, deployed applications, and possibly limited control of selecting networking components (e.g., host firewalls).

The above three are the essential models of the cloud computing but others have also emerged during the cloud computing grows.

4. Storage as a service

   The capability provided to consumers is using the storage from the remote servers as from the local storage. Some of this model provide the database storage others provide the storage used to store and retrieve any files. This model is used in almost every cloud service.

5. Information as a service

   The capability provided to consumers is a consumer can retrieve information (traffic state, stock price, weather and so on) through public API.

6. Process as a service

   The capability provided to consumers is the business process which can easily compose several different systems on your demand through the manage tools, you can use it as a system and change it agilely.

7. Integration as a service

   The capability provided to the consumer is delivering the enterprise application integration technology as the service and can be accessed anytime and anywhere.

8. Security as a service

   The capability provided to the consumer is delivering the security system through the Internet.

9. Management as a service

The capability provided to consumers is delivering the hardware and network states of the system to consumer and the consumer can use this information to manage their remote system.

10. Testing as a service

    The capability provided to consumers is delivering the service can be used to test their remote applications or local applications.

These models or we can call it cloud service patterns can be combined to provide a more sophisticated service.

The cloud computing is still in the early stage, these patterns is not the complete list, it could be changed when the industries find new ways to put their services in the cloud and find a new model to change the legacy system to cloud service.

4. Web services

4.1. Why we need web services?

Our group in the Pomodoro project is responsible to provide the web service as a cloud. Our responsibility is to offer APIs that enable developers to exploit functionality over the Internet, rather than delivering full-blown applications. So it combines the software as a service pattern, storage as a service pattern and platform as a service pattern. It provides desktop, web and mobile platforms, the possibility to put their information on a remote server which can be accessed from anywhere and anytime. It also provides a public APIs as a manual for developers.

4.2. What are web services?

Web services are typically application programming interfaces (API) or web APIs that are accessed via Hypertext Transfer Protocol (HTTP) and executed on a remote system hosting the requested services. Web services tend to fall into one of two camps: Big Web Services (WS-*s) and RESTful Web Services. [13]

Based on the definition of web service in Wikipedia, we got that web service is a protocol which allow programs written in different languages on different platforms to communicate with each other. Roy Fielding said, "The modern Web architecture emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.", according to his idea, we should focus on how the different components of different services interact, how to efficiently add new components to the system and how to improve the system's performance. So the problems fall into the integration area.

Generally there are three common styles to achieve the goal, one is plain remote procedure calls (RPC), and the second is service oriented architecture (SOA), the last is representational state transfer (REST).

5. Integration

5.1. Integration methods

Web service is designed for the communication between different organizations or between different parts of an organization, such as Enterprise Application Integration (EAI) and Cross-Organization Integration (B2B), EAI is used for integrate legacy software systems within an organization in order to allow systems to have a more complete and consistent world-view. B2B is used to allow partners and customers to interact with internal systems in a programmatic fashion. So how to solve these problems is the key for the modern enterprise use.

Basically, there are three strategies doing these integrations [14]:

First is Custom the New Protocol, this strategy is always based on exist transport level protocol such as TCP, UDP or based on the application level protocol such as HTTP, SMTP. When the new protocol is based on the application level protocols, designers usually use these protocols just for transporting messages ignoring other functionality. Then developers will study the problem domain and design a solution for particular problems. Because developers will not spend the time and money to develop the common infrastructure, it is difficult to be reused and communicate with other organizations with this new protocol. Also it is hard to design and maintain even by the experts. But there is strength for this kind of approach that it does not need to wait for any other standardization project.

Second is Building the Protocol Framework. This approach is based on the first strategy. Because different problem domains will need the different integration methods, it makes sense to establish a common framework, Pre-XML like technology COBAR and SOAP uses this method. The advantages of this strategy are that it allows developers of the framework community design the new protocol more intellect and talent and it also make the common infrastructure becomes true, it can provide the toolkit for developers who use this protocol. The obstacle of this approach is it must waiting for the standard. In general it will need a long time. This thesis will discuss this strategy in detail later.

Third is the Horizontal Protocols, instead of developing a new domain specific protocol we can use a general purpose protocol to transfer domain specific information. So we not only use the general purpose protocol as the transport method, but also use its all build in functions. Hypertext Transfer Protocol (HTTP) can be a good example. The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extensions of its request methods, error codes and headers. An important feature of HTTP is typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. HTTP allows an open-ended set of methods (“GET”, ”PUT”, ”POST”, ”DELETE”, ”OPTION”, “HEAD”) and headers that indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI), as a location (URL) or name (URN), for indicating the resource to which a method is to be applied. Messages are passed in a format similar to that used by Internet mail as defined by the Multipurpose Internet Mail Extensions (MIME) (RFC2616). [15]. As we can see from the HTTP protocol, this strategy’s most important benefit is interoperability which is not only between the applications but also between the resources. It also have some other benefits that cannot be ignored, we will discuss them in detail when introduce REST (REpresentational State Transfer) which is the best example of this strategy.

After analyzed these strategies of integration, the first Custom New Protocol can be used inside an organization but web service's goal is for the global usage, so nowadays less and less organizations use this strategy. Due to the advantages of the second strategy in the integration between organizations, it dominates the web service development for a long time. The common examples are CORBA and SOAP. Because recently more and more developers and organizations perceived the limitations of CORBA and SOAP, the third strategy gains a rapid growth. This thesis will make a detailed discussion of the last two strategies of integration through different point of views.

5.2. CORBA versus SOAP

As we mentioned above, lots of organizations want to build a framework for the integration problem, the most famous and widely used framework is CORBA and SOAP. In fact we should compare the CORBA with web services instead of SOAP, because SOAP is kind of RPC mechanism which is at the same level of CORBA. Our project is focus on the web services' integration method, so here we will give a short description between these two frameworks.

The next table shows the corresponding parts of web services and CORBA [16]:

Item	Web services/SOAP	CORBA
Protocol	SOAP, HTTP, XML Schema, JSON	IIOP, GIOP
Location identifiers	URLs	IORs, URLs
Interface specification	WSDL	IDL
Naming, directory	UDDI	Naming service, Interface Repository, Trader service

Table5.1 SOAP vs. CORBA

- Naming Issues:

  In CORBA, they use naming service, interface repository and trader service to publish services and discover each other and define how to interact over the Internet. The web services/SOAP uses the UDDI (Universal Description Discovery and Integration) to do the same job. The difference is in CORBA, each server will manage a graph of names with an initial naming context and is initially independent of any other servers, but the UDDI provides a globe service. Although CORBA can federate different organizations' naming services, it is not automatic and easy to be implemented.

- Reference issues:

  CORBA uses IORs as its object reference. It is used as an Internet-wide object reference as the URL, but the IOR is understood only by the interface repository that stores the definition of the corresponding type, which will limit the globe scalability.

- Ease to use

CORBA platform is a large and complex platform, it need installation and careful support. Web services usually use HTTP and XML which already install on almost every operating system and it is well-understood by developer.

CORBA was designed for use within a single organization or between a small numbers of collaboration organizations. It is hard to use for public interoperable environment, our project is for provide a software for public using. The result is CORBA is not a suitable integration solution for our project.

6. Representational State Transfer

The third approach of integration methods is the Horizontal Protocols. REST (REpresentational State Transfer) is an outstanding example which is first introduced by Roy Fielding in 2000. Here we will elaborate what REST is, how it works and its benefits compare with the SOAP.

REST was designed based on the HTTP/1.0 and developed concurrently with the HTTP/1.1. It is introduced by Fielding who is also the main designer of HTTP/1.1. The best well known example of REST is the WWW (World Wide Web) which is a system of interlinked hypertext documents accessed via the Internet and it also the most famous and successful Internet technology in the world.

REST is a hybrid style derived from several of the network-based architectural styles and combined with additional constraints that define a uniform connector interface. [9]

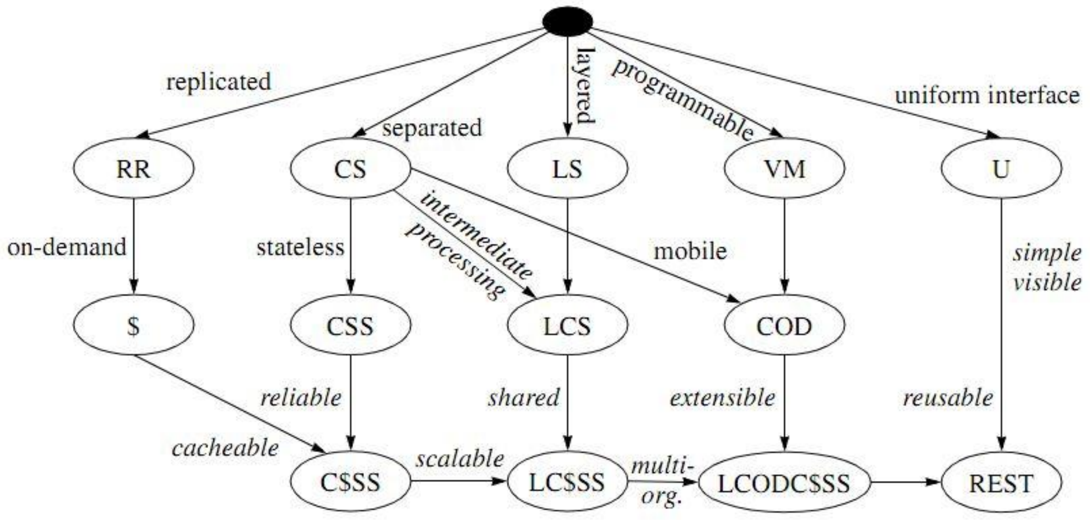


Figure6.1 REST Architectural styles

Here we will descript the each constraint and network-based architectural style based on these constraints.

1. Client-Server

   Because separating the user interface from the data storage we can improve the portability of the user interface across multiple platforms and scalability of the server components which lead to the client- server architectural style.

2. Stateless

   The communication between server and client must be stateless in nature, which lead to the client-stateless-server (CSS) style. This constraint make each request of client should have all information needed by the server which means the session state is kept by client entirely. The advantage is it improves the visibility, reliability, and scalability. In opposite, it may reduce the network performance because it need send repetitive information. But in the current Internet environment the bandwidth is good enough to handle this obstacle.

3. Cache

   The advantage of this constraint is they may partially or completely eliminate some interactions, improve efficiency, scalability and user-perceived performance. To achieve it, response should implicitly or explicitly label cacheable or non-cacheable

4. Uniform Interface

   Uniform interface means using the general interface between components. The advantages are, first, it makes intermediaries know more about the interactions between components which makes securing the protocol possible and extending/enhancing anticipated protocol semantics possible. Second, it improves scalability because interfaces stay static. Third, a small and fixed set of semantics lowers the cost of coordination between uncoordinated actors. Fourth, it improves the simplicity and visibility of interaction. There are four interface constraints which are used to obtain a uniform interface: identification of resources; manipulation of resources through representations; self-descriptive message; and, hypermedia as the engine of application state.

5. Layered system

   Layered system constraint is used popularly. It will further improve behavior for Internet-scale requirement. By separating the system into hierarchical layers, it is easy to encapsulate legacy services and move simplifying components to intermediary. Also intermediary can be used to improve system scalability by enabling load balancing of services across multiple networks and processors.

6. Code-On-Demand

   REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. It will simplify clients' implementation but will deduce the visibility, so it is the optional constraint.

These constraints make REST as the common integration architecture. It not only used in the web service but also can be used in other distributed system. But this thesis will focus on how it works in web services environment. Because it is derived concurrently with HTTP1.1, HTTP1.1 does very well on the criteria of REST. Next we will introduce the resource-oriented architecture which follows these constraints, how it works, how to implement it.

7. Resource-Oriented Architecture

7.1. Resource Oriented Model

The Resource Oriented Model focuses on those aspects of the architecture that relate to resources. Resources are a fundamental concept that underpins much of the Web and much of Web services; for example, a Web service is a particular kind of resource that is important to this architecture.

The ROM focuses on the key features of resources that are relevant to the concept of resource, independent of the role the resource has in the context of Web services. Thus we focus on issues such as the ownership of resources, policies associated with resources and so on. Then, by virtue of the fact that Web services are resources, these properties are inherited by Web services. [17]

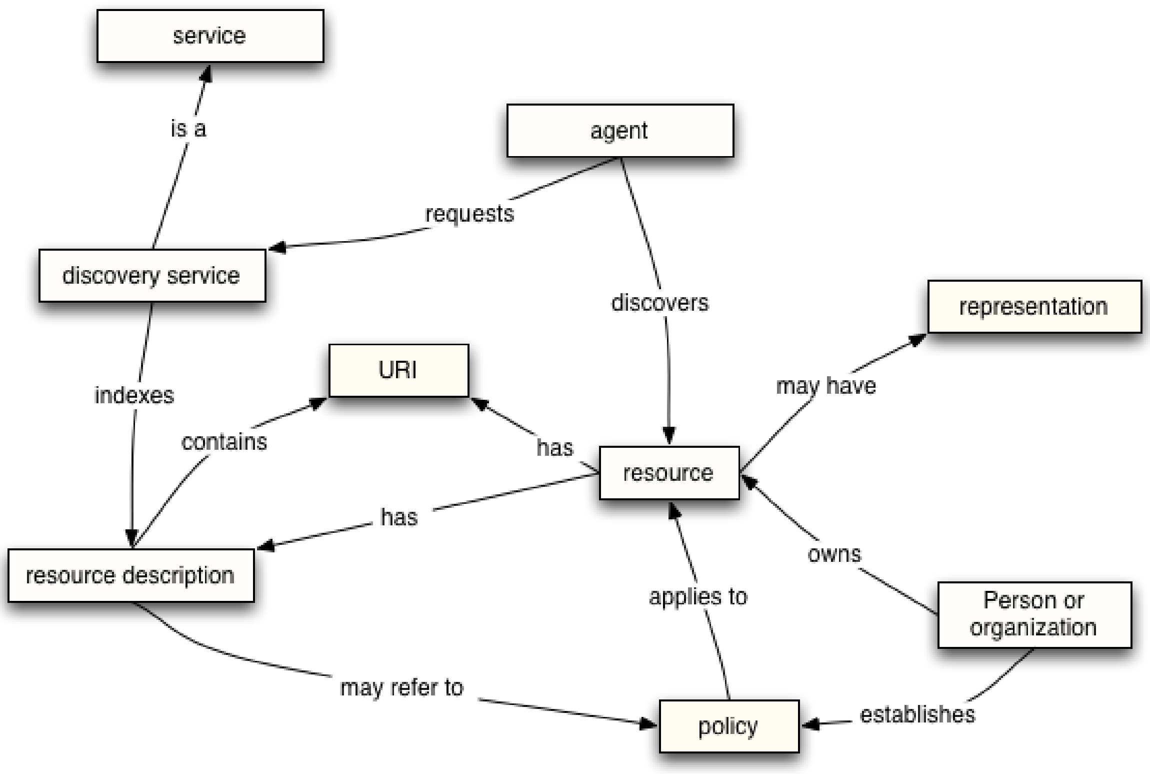


Figure 7.1 Resource Oriented Model

7.2. Resource-Oriented Architecture

Due to the description of resource-oriented model, we know it is basically coupled with web. REST is a general design criterion of architecture for integration. It can be used based on many platforms. But the most suitable one is Web because of REST's history. Resource-oriented architecture is a specific set of guidelines of an implementation of the REST architecture based on the web platform.

7.2.1. Resource definition

The resource is anything that your system wants to expose, if your system "wants to create a hypertext link to it, make or refute assertions about it, retrieve or cache a representation of it, include all or part of it by reference into another representation, annotate it, or perform other operations on it.”[18] In common web service usually exposes the following kinds of resources:

1. Predefined one-off resources

   This kind of resource usually is used as top-level directories such as the web site’s home page which contains all the resources provided by this service. It is important for REST’s constraint Uniform interface which contains one of four interfaces constraints called hypermedia as the engine of application state. This concept will introduce it later in this chapter.

2. Individual items of data

   Any kind of object can be a resource to expose.

3. The result of algorithms

   This kind of resource could be the output of a specific algorithm or could include collection resources based on some criteria which usually the results of queries. Sometime this kind of resources’ number could be infinite.

7.2.2. Uniform Resource Identifier

The uniform resource identifier has been successfully deployed as means to share information since the creation of the web. There are lots of substantial benefits using URIs which are linking, bookmarking, caching, and indexing by search engines. Due to the advantage of it, it could be a perfect solution to identify a resource. Each resource of a service must have at least one distinct URI address. For example:

Resources	URIs
All activities of a user	http://demo.pomodoroproject.net/Pomodoro/api/*{userID}*/activities/
All project of a user	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/

7.2.3. The Uniform Interface

Although each resource has its own identifier, how to use it become a question. As the description in HTTP1.1 specification, every resource supports the common interface or called common method: GET, POST, PUT, DELETE, HEAD, and OPTION.

These methods are used to communicate between client and server, the client could use these methods to retrieve and manipulate the resource on the server through Internet, the implementer of the service should be careful that the client might take unexpected action on server. In particular, the convention has been established that the GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. These methods ought to be considered "safe". This allows user agents to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested. [19]

Some methods also could have the property of idempotence which can be called idempotent method. Idempotence method means the side-effects of N > 0 identical requests is the same as for a single request. The method GET, HEAD, PUT, DELETE and OPTIONS are idempotence method, which means it can resend several times but without side effect.

Safety and idempotence are two very important properties in these methods. Because it let a client could make reliable HTTP requests over an unreliable network.

Next we will give a detail description of each method and how to use it in the resource-oriented architecture.

HTTP GET method is used to retrieve the resource presentation identified by the URI, the most useful situation in this method is it support the "conditional GET" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. It is the crucial technic for the service can use the cache to improve the performance.

HTTP PUT method is used to modify the resource identified by the URI, and the request usually contain the entity body descript the new information about this specific resource. The state code in response can be used to check the operation's state.

HTTP DELETE method is used to delete the resource identified by the URI. As the PUT method, It can also use the state code to decide whether delete operation success or not.

HTTP POST is the most complicated method because it can be used in many situations, but the most common case is creates a new resource under the given URI. As described in RFC2616 about this method: POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

As a result, the POST method's actual function is determined by the server and dependent on the request-URI.

HTTP HEAD is used for retrieve a metadata-only representation and HTTP OPTIONS is used to check which HTTP methods a particular resource supports.

7.2.4. Representations

A representation is data that encodes information about resource state [18]. Representations of a resource may be sent or received using interaction protocols. These protocols in turn determine the form in which representations are conveyed on the Web. HTTP, for example, provides for transmission of representations as octet streams typed using Internet media types.

After the system construct the appreciated URI for each resource, there is a problem arise. The resource is not the data. The resource is determined by the system designer, so it cannot send directly. The system only could send a series of bytes about the current state of a resource, in a specific file format, in a specific language, which is called the representation.

There are lots of data format can be used to represent the resource, the most used are XML, JSON and their variant. XML is document oriented and JSON is data oriented, they can use in different situation depend on your system.

7.2.5. Hypermedia as the Engine of Application State

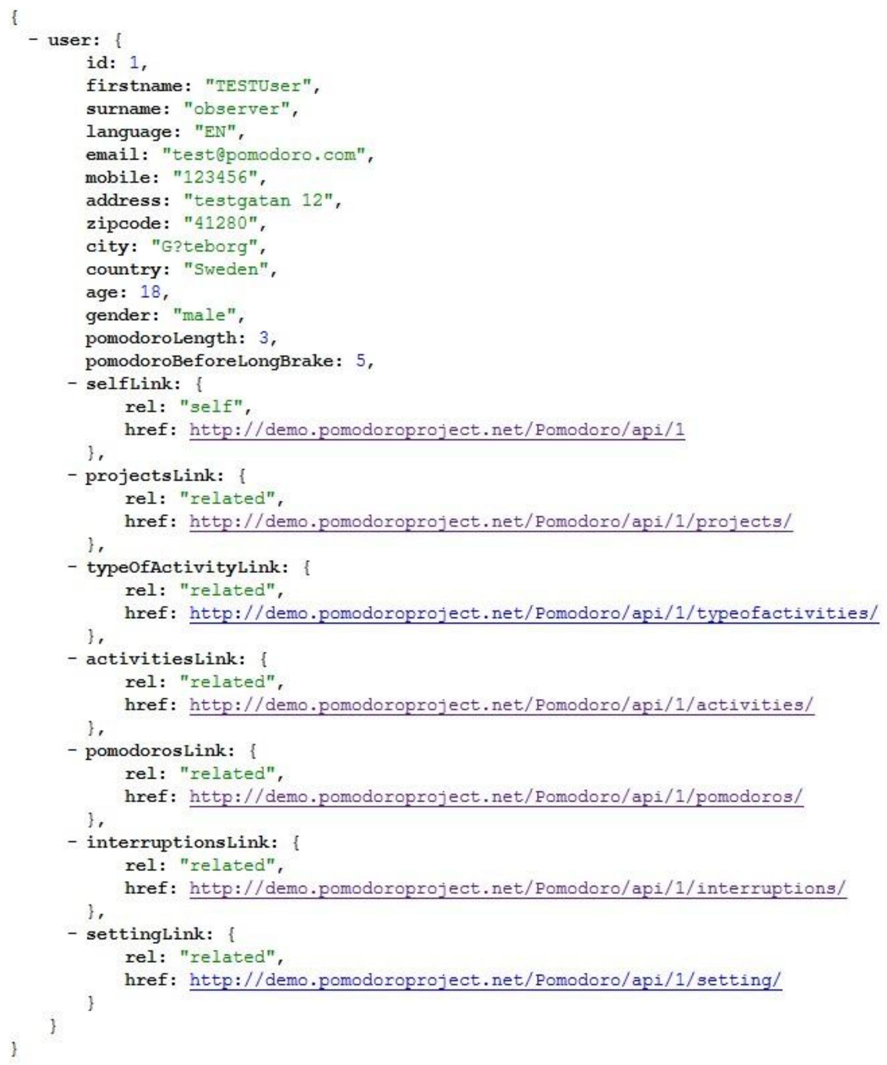
In order to develop a system that works in harmony with the Web, one needs to carefully model distributed application state, business processes that affect that state, distributed data structures which hold it, and protocols that drive interactions between the different parts of the system. HATEOAS (hypermedia as the engine of application state) is a design pattern that can greatly help building software to meet these demands.

HATEOAS (hypermedia as the engine of application state) refers to the use of hyperlinks in resource representations as a way of navigating the state machine of an application.
An application uses a set of interactions as an application protocol to achieve its goal. The application state is a snapshot of the execution of such an application protocol. It defines the interaction rules in a system. So the application state is also the snapshot of the system.
According to the application protocol each resource representation contains links which is transferred between the participants. Each link in the representation advertises one resource of the system, for example, when the client submits the initial request to the system:

Request:

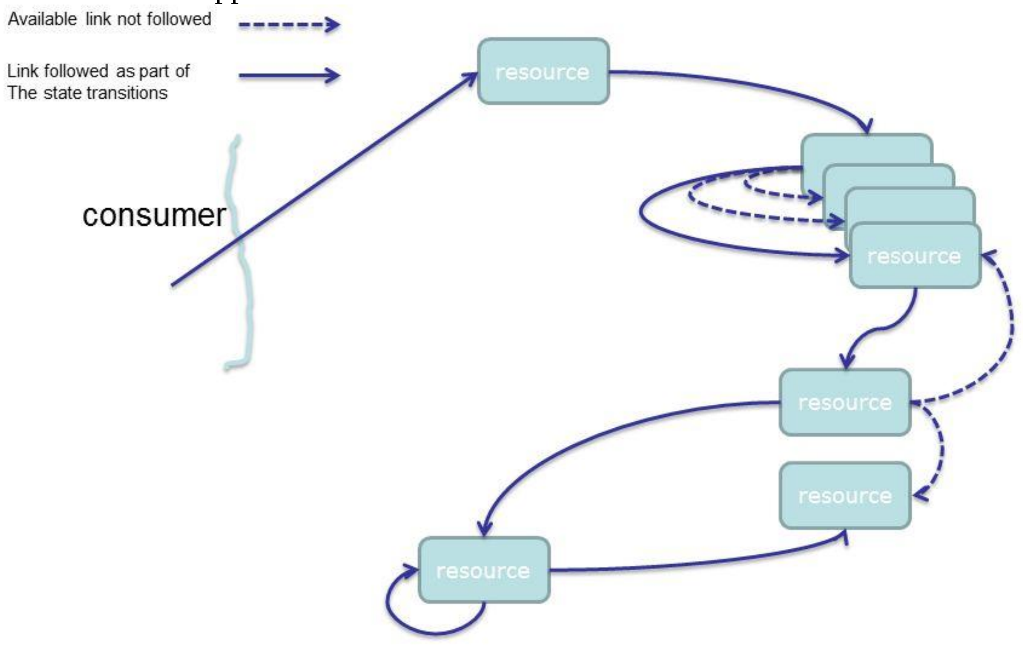
```
GET /Pomodoro/api/1 HTTP/1.1
```

Response:



```
{
  - user: {
        id: 1,
        firstname: "TESTUser",
        surname: "observer",
        language: "EN",
        email: "test@pomodoro.com",
        mobile: "123456",
        address: "testgatan 12",
        zipcode: "41280",
        city: "G?teborg",
        country: "Sweden",
        age: 18,
        gender: "male",
        pomodoroLength: 3,
        pomodoroBeforeLongBrake: 5,
      - selfLink: {
            rel: "self",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1
        },
      - projectsLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/projects/
        },
      - typeOfActivityLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/typeofactivities/
        },
      - activitiesLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/activities/
        },
      - pomodorosLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/pomodoros/
        },
      - interruptionsLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/interruptions/
        },
      - settingLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/setting/
        }
    }
}
```

We can find that the system's all father resource is contained by this initial request response. The client could choose the links and interact with server with this specific URI in order to transition to next application state.



Available link not followed
Link followed as part of The state transitions
consumer
resource
resource
resource
resource
resource

7.2.6. Self-Descriptive message

Self-descriptive messages describe how we view and interact with resources. First, the message should be able to easily examine a representation of a resource to understand the structure and content, and use this information to intelligently manipulate the resource. Second, the message should also contain the extra information about the resource such as cache ability, attribution information. For example:

```
GET /api/1 HTTP/1.1
User-Agent: SomeClient
Host: pomodoroproject.net
Accept: json/application
If-None-Match: "234902340932423432"
```

If we follow the RFC2616 to construct the message, it will be the self-descriptive message and the intermediate will know how to handle it.

7.2.7. Addressability

An application's addressability is a very important feature. In ROA, resources expose themselves with the help of the URIs. According to this, client can easily overlook and work with any piece of resource they want.

7.2.8. Statelessness

The second important feature of ROA is statelessness which means every HTTP request happens in complete isolation [20]. Each request possesses all of the necessary information for server. So clients keep track of any state information. This simplifies the server needs, which can result in easier scalability and performance enhancements. With statelessness, the server never has to worry about the client; it is much easier to distribute an application across load-balanced servers; it is easy to cache. For example, with statelessness, it isolates the client against changes on the server as it is not dependent on talking to the same server in two consecutive requests. A client could receive a document containing links from the server, and while it does some processing, the server could be shut down, its hard disk could be ripped out and be replaced, the software could be updated and restarted — and if the client follows one of the links it has received from the server, it won't notice.

The ROA's benefit is almost inherited from HTTP, it is successful used all over the world, why not use this great and mature platform to do the integration of applications.

7.3. ROA versus WS-*

In general, ROA's most important benefit is it is simpler than WS-* and other integration methods. It used the well-known technology as the foundation. It is easily to understand and just need a short learning curve which also very important in the enterprise environment. Here we will give a roughly comparison based on the non-functional requirement.

- Network performance: WS-*: No explicit focus on coarse grained (document oriented) messages, accidental design towards fine grained or control messages can degrade network performance. REST: Coarse-grained, document oriented messages are encouraged. And SOAP using the XML wrapper around every request and response. But in REST, every request needs to send all necessary information which will reduce the performance.

- Network Efficiency: WS-*: The lack of any visibility completely removes the ability to use caching. Responses should always be marked as non-cacheable to avoid that existing HTTP caches negatively impact the application. REST: Visibility enables caching which improves network efficiency.

- Visibility: WS-*: No visibility because meaning of message depends on understanding action. The effect on trust will usually be that the filtering party blocks all requests to the single service endpoint. REST: Messages are self-describing and can be understood by intermediaries. This is the foundation for caching and trust (firewall admins can understand the messages and allow exactly the traffic they want to let through).

- Evolvability: WS-*: Tight coupling of client and server prohibits independent evolution. REST: Coupling between client and server is removed, server owners need not know about client particularities to evolve the servers without breaking clients.

- Simplicity: REST: Maximized by uniformity of connectors and identifiers. The web service can easily be accessed by the URI which is already well know and widely used. The developer can easily create and modify the URI to access the different web resources. The response also is simple. For example the POX, JSON, the developer can read the response and phase it.

- Scalability: WS-*: Server statelessness not guaranteed by a constraint. Scalability depends on application design. REST: Message self-descriptiveness constraint mandates message statelessness which in turn mandates the stateless server constraint. The latter guarantees scalability.

- Security: WS-*: WS-Reliability and WS- Security almost guaranteed the security, but lacking of visibility make it difficult to set up the peripheral firewall. REST: REST calls could go over HTTPS, the administrator (or firewall) can discern the intent of each message by analyzing the HTTP command used in the request.

8. Data Representation format

The choice of an adequate data representation format can have significant consequences on data transmission rates and performance. So it is a key architectural decision following integration method. Naturally, SOAP only uses XML as its data representation format and REST can use numerous MIME (Internet media type) types such as JSON. JSON and XML are most popular representation data formats being used today. Following that, our research focuses on JSON and XML.

The Extensible Markup Language (XML) is a subset of the Standard Generalized Markup Language (SGML) and evolved as a result of the complexity of SGML. The intent of an XML document is self-evident and embedded in its structure. The fundamental design considerations of XML include simplicity and human readability. Amongst the design goals of XML, the W3C specifies that *"XML shall be straightforwardly usable over the Internet"* and *"XML documents should be human-legible and reasonably clear."* [21]

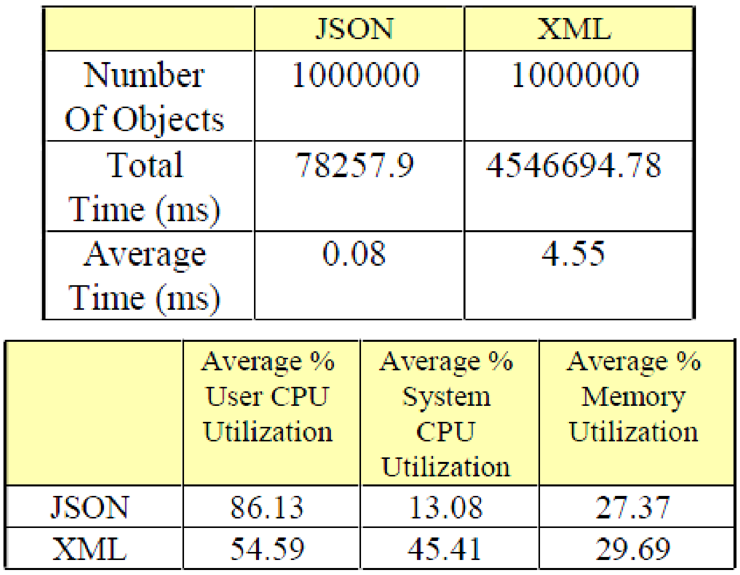
The primary uses for XML are Remote Procedure Calls (RPC) and object serialization for transfer of data between applications. XML is a language used for creating user-defined markups to documents and encoding schemes. XML does not have predefined tag sets and each valid tag is defined by either a user or through another automated scheme. Vast numbers of tutorials and user forums provide wide support for XML and have helped create a broad user base. XML is a user-defined hierarchical data format. An example of an object encoded in XML is provided following

```
<address>
        <country>Sweden</country>
        <city>Gothenburg</city>
        <street>Uppstigen</street>
</address>
```

JSON is designed to be a data exchange language which is human readable and easy for computers to parse and use. JSON is directly supported inside JavaScript and is best suited for JavaScript applications; thus providing significant performance gains over XML, which requires extra libraries to retrieve data from Document Object Model (DOM) objects. JSON is estimated to parse up to one hundred times faster than XML in modern browsers, but despite its claims of noteworthy performance, arguments against JSON include lack of namespace support, lack of input validation and extensibility drawbacks. The code following describes an example where JSON is used to encode an *address* object.

```
{
        address : {
                "country": "Sweden",
                "city" : "Gothenburg",
                "street" : "Uppstigen"
        }
}
```

In light of the research of comparison the differences of two current data representation formats (JSON and XML) of Nurzhan Nurseitov et al. [22], the result indicates that JSON is faster and uses fewer resources than its XML counterpart as showed following:



	JSON	XML
Number Of Objects	1000000	1000000
Total Time (ms)	78257.9	4546694.78
Average Time (ms)	0.08	4.55

	Average % User CPU Utilization	Average % System CPU Utilization	Average % Memory Utilization
JSON	86.13	13.08	27.37
XML	54.59	45.41	29.69

Figure 8-1 Performance comparison between XML and JSON

Overall, JSON is data-structure-based data representation format and XML is document-based. According to the Pomodoro project requirements - the structure of data set is not complicated, plus JSON has a great performance advantage, finally we chose JSON as data representation format.

9. Authentication & Authorization Methods

Security solution is one of main concerns of the project as well. Security is an absolute need in today's software applications. Since the trend is to use web-based software, thanks to cloud computing, new security issues arise. Software does not run anymore in a small and manageable environment but rather in an environment with many uncertainties concerning the users, the participating parties and systems. Thus, the application itself must address security and provide adequate mechanisms.

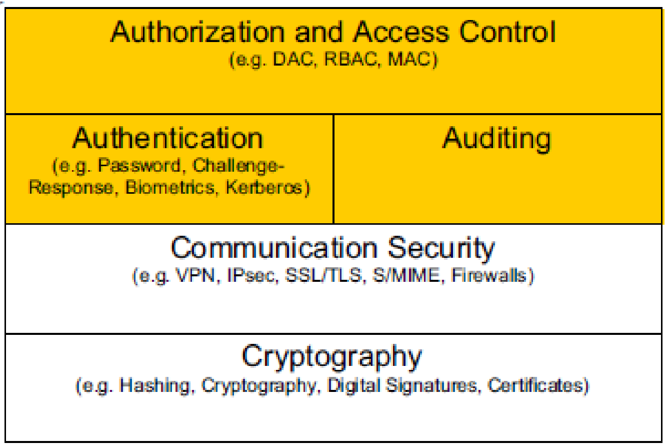

Authorization and Access Control
(e.g. DAC, RBAC, MAC)
Authentication
(e.g. Password, Challenge-Response, Biometrics, Kerberos)
Auditing
Communication Security
(e.g. VPN, IPsec, SSL/TLS, S/MIME, Firewalls)
Cryptography
(e.g. Hashing, Cryptography, Digital Signatures, Certificates)

Figure 9-1 Level of security mechanisms [23]

As showed in figure 9-1, low-level security addresses the secure transmission of data through an untrusted networking, using cryptography like digital signatures and communication security mechanisms like SSL/TLS. At high level, the security is intended to protect the application itself by defining a security model. R. Sandhu et al. (1996) presented the model that comprises several mechanisms to enforce the desired security policy.

- Authentication: establishes the identity of one party to another. Thus, authentication needs to prove the identity of a certain user to the system.
- Access Control: determines whether a user (subject) is allowed to access an object or not. This decision is based on the authorization of the system wide security policy.
- Auditing: gathers data about activity in the system and analyzes it to discover security violations and diagnose their cause.

Our research concentrates on the how to provide authorization solution in a cloud-based 3-leggged web services environment. And for authentication, the data of the project is all private that makes solution for authentication relatively simple. So we decide to implement identity management by ourselves.

Traditional authorization method (2-legged) is not suited for the project. With the increasing use of distributed web services and cloud computing, third-party applications (consumers) require access to server (services provider) -hosted resources (3-legged). These resources are usually protected and require authentication using the resource owner's credentials (typically a username and password). In the traditional client-server authentication model, a client accessing a protected resource on a server presents the resource owner's credentials in order to authenticate and gain access. For 3-legged web application, Resource owners (user) should not be required to share their credentials when granting third-party applications (consumers) access to their protected resources hosted by servers (service providers). They should also have the ability to restrict access to a limited subset of the resources they control, to limit access duration, or to limit access to the methods supported by these resources. Following figures show the differences between these two.

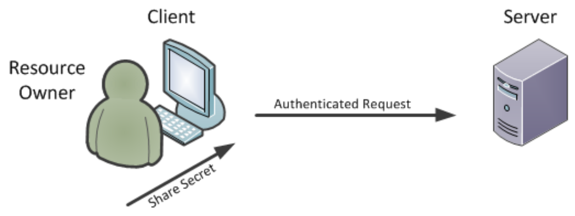


Figure 4-3 traditional web application

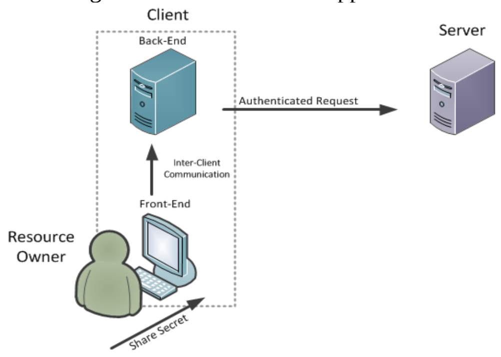
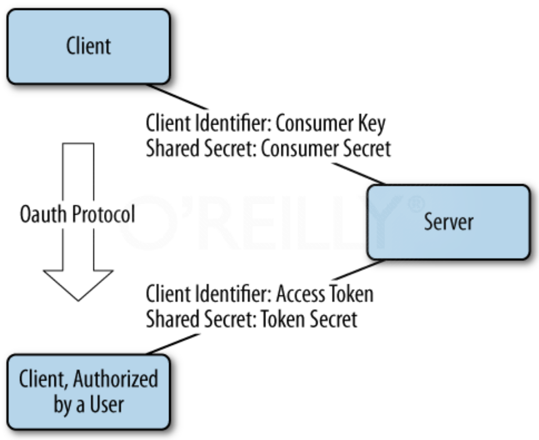


Figure 9-2 3-legged web application

For handling 3-legged web application authorization, actually there are several choices including OAuth, AuthSub(Google), OpenAuth(AOL). According our study, OAuth has been widely adopted by industrial. Google, Facebook and Twitter have applied OAuth as their basic authorization method.

OAuth began in November 2006, at the 73rd Internet Engineering Task Force (IETF) meeting in Minneapolis in November of 2008, an OAuth BOF was held to discuss bringing the protocol into the IETF for further standardization work. On 2009-04-23 a security flaw in the 1.0 protocol was announced. It affects the OAuth authorization flow (also known as '3-legged OAuth') in OAuth Core 1.0 Section 6.[24] So our implementation of OAuth authorization protocol follows OAuth 2.0 version whose specification is being developed within the IETF OAuth WG and is expected to be finalized by the end of 2010.

OAuth provides a method for making authenticated HTTP requests using a token – an identifier used to denote an access grant with specific scope, duration, and other attributes. Tokens are issued to third-party clients by an authorization server with the approval of the resource owner. OAuth defines multiple flows for obtaining a token to support a wide range of client types and user experience. In other words, The OAuth protocol enables a website or application (known as a *service consumer*) to access protected resources from a web service (known as a *service provider*) through an API. The API does not require users to disclose their service provider credentials to consumers. For example, a web user (resource owner) can grant a printing service (client) access to his/her protected photos stored at a photo sharing service (resource server), without sharing her username and password with the printing service. Instead, he/she authenticates directly with the photo sharing service (authorization server) which issues the printing service delegation-specific credentials (token). Following figure shows how OAuth works with 3-legged web application on high level:


Client
Client Identifier: Consumer Key
Shared Secret: Consumer Secret
Oauth Protocol
Server
Client Identifier: Access Token
Shared Secret: Token Secret
Client, Authorized by a User
Figure 9-3 OAuth works with 3-legged web application

As figure 4-6 showed, OAuth protocol specifies following steps:

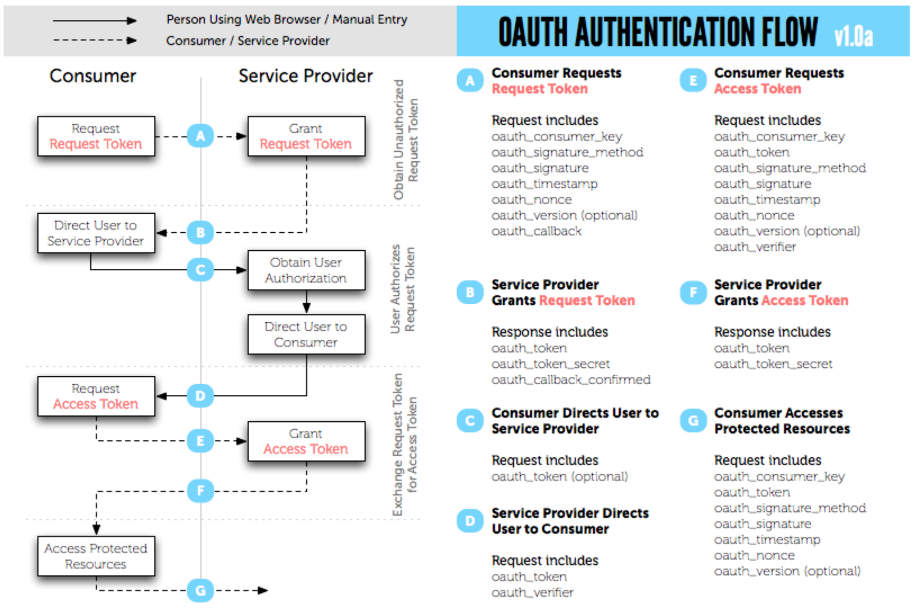


Figure 9-4 OAuth work flow

According to reported study above, we, Pomodoro service provider, chose OAuth to be the protocol to deal with 3-legged web application authorization issues.

10. RESTful web service Frameworks

Unlike WS-*, REST is not a mature industry standard, so we have few frameworks that can support REST development. During our evaluation, we mainly focused on three frameworks including:

- Restlet: Restlet is a lightweight, comprehensive, open source REST framework for the Java platform. Restlet is suitable for both server and client Web applications. It supports major Internet transport, data format, and service description standards like HTTP and HTTPS, SMTP, XML, JSON, Atom, and WADL. A GWT port is also available. The Restlet framework is composed of two main parts. First, there is the "Restlet API", a neutral API supporting the concepts of REST and facilitating the handling of calls for both client-side and server-side applications. This API must be supported by a Restlet implementation before it can effectively be used. Multiple implementations could be provided (open source projects or commercial products).

- RESTEasy: RESTEasy is a JBoss project that provides various frameworks to help you build RESTful Web Services and RESTful Java applications. It is a fully certified and portable implementation of the JAX-RS specification. JAX-RS is a new JCP specification that provides a Java API for RESTful Web Services over the HTTP protocol.

- Jersey: Project Jersey is an open source community that is building the production quality reference implementation of JSR-311: JAX-RS - Java API for RESTful Web Services. Jersey implements support for the annotations defined in JSR-311, making it easy for developers to build RESTful web services with Java and the Java JVM. Besides implementing the JSR-311 API, Jersey provides an additional API not specified by JSR-311 so that developers can extend this JSR to suit their specific needs.

During the implementation phase, we separately implemented simple cases using 4 standard HTTP methods (GET, POST, PUT, and DELECT) to manipulate data presentation for evaluating the three frameworks mentioned above. We found that they all have advantages and disadvantages such as RESTEasy have a good document support but don’t have support for OAuth, and Restlet source code is easy to study and understand but it is supported by relative small community (few people), and there is almost no update for long time. Finally we chose Jersey including following considerations:

- Implementation of JSR-311 (JAX-RS: Java API for RESTful Web Services)
- Supported by good community (Java community)
- Open source
- Good documentation
- Good support for OAuth authorization protocol

11. Design and Implementation

11.1. Design

11.1.1. Domain model

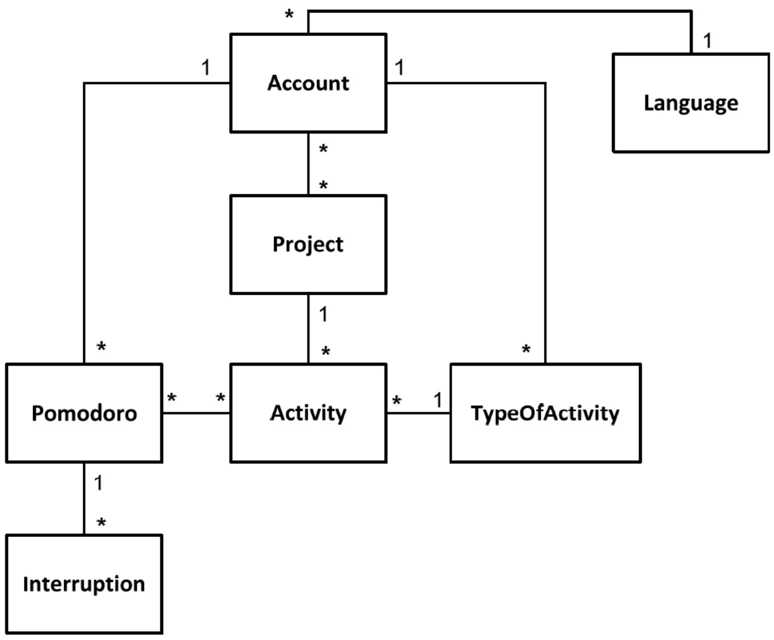


Figure 11.1 Pomodoro domain model

- Account: The account object represents a user and the users' information.
- Project: A project act as a container for activities, each project contains a set of activities. Project is similar to a workspace or group where several users can share activities.
- Activity: The activities that are selected when executing a Pomodoro.
- TypeOfActivity: User defined types for activities.
- Pomodoro: The time box representation that are connected to a user and one or several activities.
- Interruption: Interruptions, internal or external, that occurs during a Pomodoro.
- Language: User selected language for the interface.

11.1.2. Database design

Our database design was following domain model. Each object and each relationship of objects are presented in a relational database. Following is Entity-Relationship diagram:

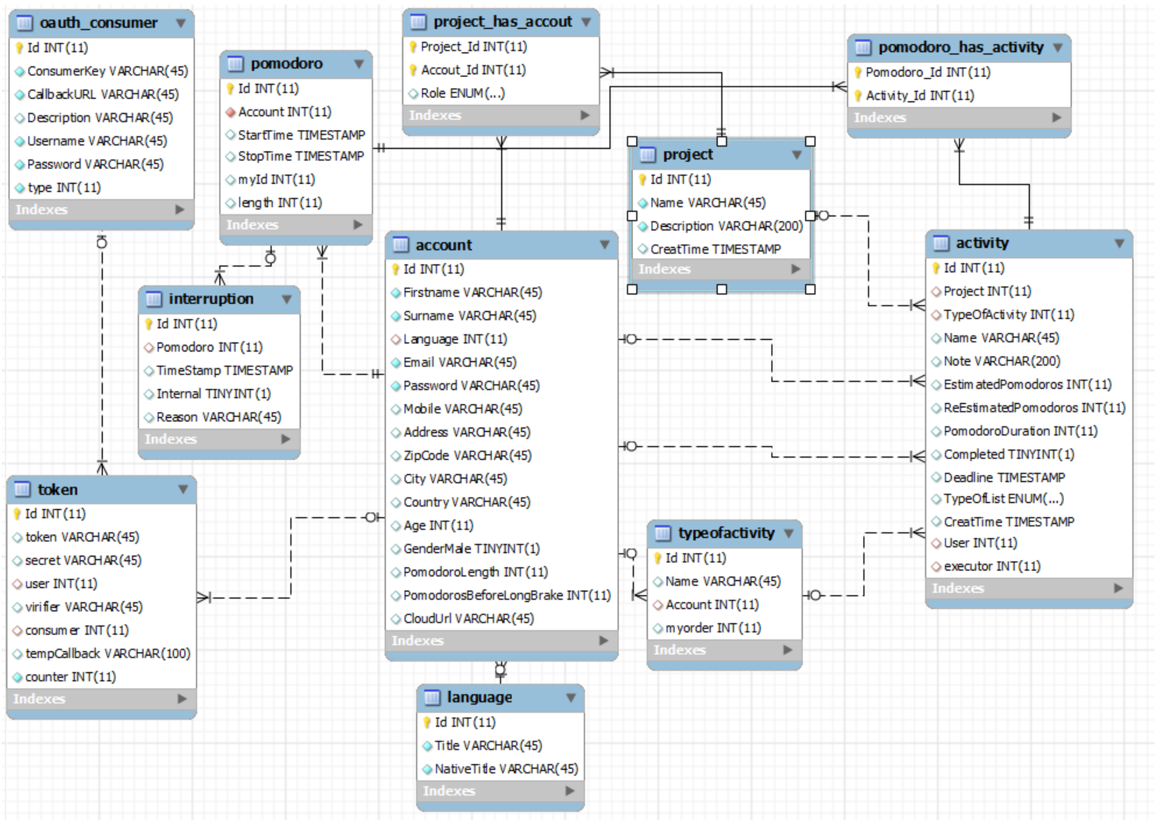


Figure 11.2 Pomodoro project E-R diagram

As you have noticed in the E-R diagram, besides the objects of domain model, there are two additional entities called *OAuth_consumer* and *token*, these two is specially for implementing OAuth. Since RESTful web service inherits the stateless constraint of REST architectural style, we need to record authorization information in the database for further use instead of storing in session in which way traditional web application dose.

11.1.3. Functional requirements

Execute a Pomodoro

- 1) Define Activity for the current Pomodoro
- 2) Start the Pomodoro timer, set to default 25 minutes
- 3) Alert user when Pomodoro timer reaches 0
- 4) Stop the Pomodoro timer, a break count up starts
- 5) Mark Activity as done
- 6) Archive Activity when marked as done
- 7) Inform user every 4 Pomodoro for a longer break

Activity lists

- 8) CRUD Activity to To-do Today list
- 9) CRUD Activity to Activity Inventory list
- 10) CRUD Activity to Unplanned & Urgent list
- 11) View To Do Today list
- 12) View Activity Inventory list
- 13) View Unplanned & Urgent list
- 14) Move Activity from To-do Today list to Activity Inventory
- 15) Move Activity from Activity Inventory to To-do Today

- 16) Move Activity from Unplanned & Urgent list to To-do Today or Activity Inventory
- 17) Sort on columns for To Do Today and Activity Inventory List

Connect to cloud

- 18) Authenticate user with the cloud
- 19) CRUD operations in the cloud
- 20) Create and read Pomodoro Statistics in the cloud

Records and estimations

- 21) Set estimated Pomodoros for an Activity
- 22) Track number of Pomodoros executed for an Activity

Interruptions

- 23) Note an internal interruption, increase the value for the current Pomodoro
- 24) Note an external interruption, increase the value for the current Pomodoro
- 25) Create Activity to Unplanned & Urgent list

Settings

- 26) Change the length of a Pomodoro
- 27) Change the length of the break between Pomodoros
- 28) Change the number of Pomodoros before a longer brake
- 29) Change login information for the cloud service

Next section we will introduce the use cases design according to functional requirements mentioned here.

11.1.4. Use cases design

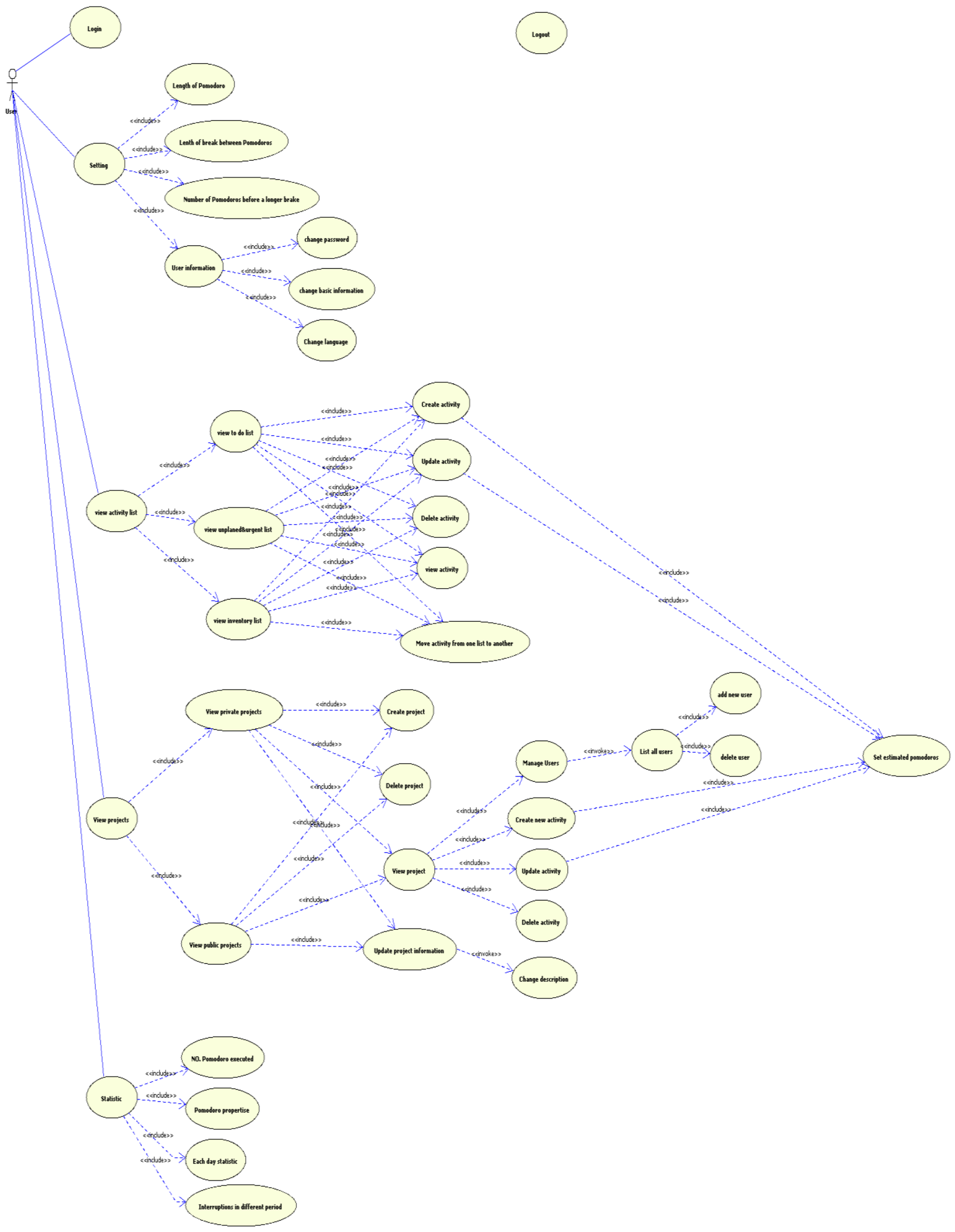


Figure 11.3 use case design

11.1.5. RESTful web services Design principle

Our design principle follows procedure introduced by Leonard Richardson and Sam Ruby (2007) in their book RESTful Web Services [20]:

1. Figure out the data set: This is the data set you'll be exposing and/or getting your users to build, in our case, Pomodoro technique would be exposing in different services to consumers.

2. Split the data set into resources: Then we need to decide how to expose the data as HTTP resources. A resource is anything interesting enough to be the target of a hypertext link. For example, in our case a Pomodoro is kind of resource web service would expose.

For each kind of resource:

3. Name the resources with URIs: After identifying resources, each of them needs a unique name which can be used to locate the resource in the Internet. RESTful web service use URI to be the identifier of resource.

4. Expose a subset of the uniform interface: When user can find the resources exposed by web services using URI you need to tell users what they can do. In RESTful web services; basically, we use 4 stand HTTP methods as uniform interfaces: GET, POST, PUT, DELET. We use GET to acquire resources, POST to add new resources, PUT to update existed resources and DELET to remove resources.

5. Design the representation(s) accepted from and served to the client: Just as the name implies, REpresentational State Transfer, in REST we transfer representation to change the state of client. In other words, we use representation of resource to operate this resource. So design the representation of resource is important part of implementation of RESTful web services. Here we need to consider which data inter-change format we will use, for example JSON & XML, then how to use the selected data inter-change format to represent the resource (data) and what form of data inter-change format we can accept from the user.

6. Integrate this resource into existing resources, using hypermedia links and forms: Link to other resources is referred to Hypermedia as the Engine of Application State (HATEOAS), which is key constraint of REST.

"*REST APIs must be hypertext-driven*"

Roy T. Fielding, 2008 [25]

After design the presentation of a resource we need to consider all the related resources and list them following the original resource presentation.

7. Consider the typical course of events: what's supposed to happen? It means what will happen when server gets a usual HTTP request? We need to decide which numeric response code the response will have, and what HTTP headers and/or entity-body will be provided. We also need to consider how the request will affect

resource state. For example, when we successfully finish an operation we will send HTTP code 200 (“OK”) to client who sent the request.

8. Consider error conditions: what might go wrong? A request that creates, modifies, or deletes a resource has more failure conditions than one that just retrieves a representation. Even a simple GET method that retrieves a representation also has unpredictable failure condition. For example, server cannot connect to database. In many cases, HTTP code is not enough; we need to give the consumer who are using our web service more specific information so that consumer can handle the failure much easier.

11.1.6. High-level Pomodoro web service architecture design

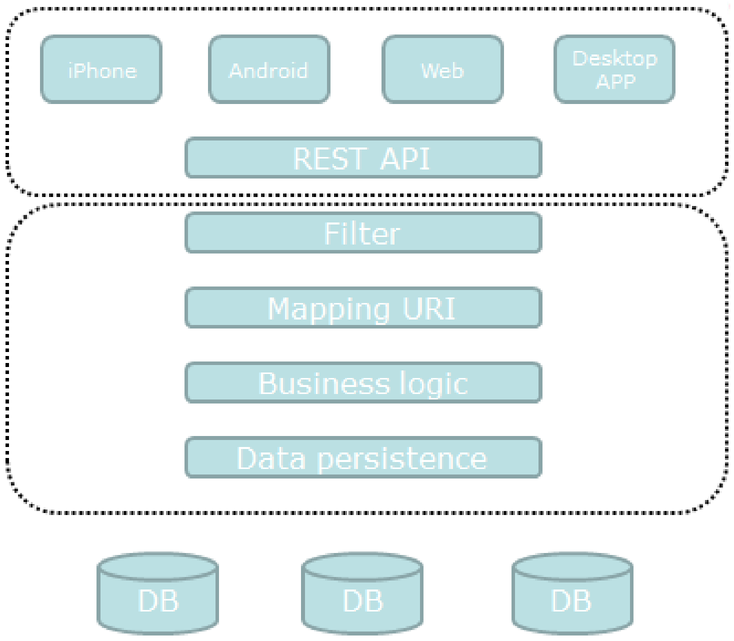


Figure 11.4 High-level architecture of Pomodoro web service

The high-level architecture design is based on study of REST architectural style, OAuth and the designs mentioned above. By dividing project into different layers, in which each layer concentrates on their own job, we can have loose-coupling system that is flexibility, maintainability. For example, we implemented business logic by ourselves, but in fact we framework now is supporting RESTful web services and gives us better solution. So if we want to use Spring framework which is better well-designed than what we have done in our business logic layer of the project, we can do that very easy. Moreover, the implementation needs to be fulfilled the constraints of REST such as stateless, cacheable, and so on.

REST API: On the top level, this layer is responsible for integrating with iPhone Android mobile application, web application and desktop application using JSON as data inter-change format. URLs are our open APIs.

Filter: We implemented OAuth protocol in this layer. The task of this layer is to filter request using OAuth protocol. Those requests that have not passed the OAuth authorization will be denied in this layer.

Mapping URI: After filtering request, we need to match those validated requests to corresponding resources. We chose Jersey (JAVA RS-311) framework that helps us to build RESTful API easier and efficiently.

Business logic: In this layer, we combined original data set, and then we use JSON to represent the resource, make it ready to be exposed. This layer also handles the received the JSON and parse the JSON into usual built-in data structure of programming language, in our case, Java.

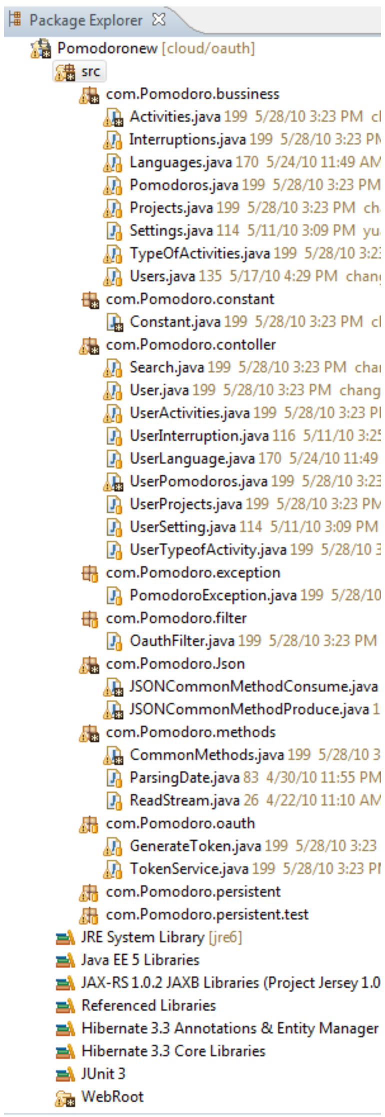
Data Persistence: Persistence deals the interaction with database. In the layer, we use Hibernate to handle data persistence.

Following we will go detailed to introduce how we implemented the Pomodoro web services in action taking part according to the designs mentioned in action planning phase above.

11.2. Implementation

In this section, firstly, we would like to introduce the implementation of architecture on the programming level. Then we will present the implementation of the project based on the RESTful web services design principle mentioned in action planning phase. And then the implementation of OAuth protocol will be introduced.

11.2.1. Overview of the project



```
Package Explorer
  Pomodoronew [cloud/oauth]
    src
      com.Pomodoro.bussiness
        Activities.java 199 5/28/10 3:23 PM cl
        Interruptions.java 199 5/28/10 3:23 PM
        Languages.java 170 5/24/10 11:49 AM
        Pomodoros.java 199 5/28/10 3:23 PM
        Projects.java 199 5/28/10 3:23 PM ch
        Settings.java 114 5/11/10 3:09 PM yu
        TypeOfActivities.java 199 5/28/10 3:2
        Users.java 135 5/17/10 4:29 PM chan
      com.Pomodoro.constant
        Constant.java 199 5/28/10 3:23 PM cl
      com.Pomodoro.contoller
        Search.java 199 5/28/10 3:23 PM cha
        User.java 199 5/28/10 3:23 PM chang
        UserActivities.java 199 5/28/10 3:23 PI
        UserInterruption.java 116 5/11/10 3:2
        UserLanguage.java 170 5/24/10 11:49
        UserPomodoros.java 199 5/28/10 3:2
        UserProjects.java 199 5/28/10 3:23 PM
        UserSetting.java 114 5/11/10 3:09 PM
        UserTypeofActivity.java 199 5/28/10 3
      com.Pomodoro.exception
        PomodoroException.java 199 5/28/10
      com.Pomodoro.filter
        OauthFilter.java 199 5/28/10 3:23 PM
      com.Pomodoro.Json
        JSONCommonMethodConsume.java
        JSONCommonMethodProduce.java 1
      com.Pomodoro.methods
        CommonMethods.java 199 5/28/10 3
        ParsingDate.java 83 4/30/10 11:55 PM
        ReadStream.java 26 4/22/10 11:10 AM
      com.Pomodoro.oauth
        GenerateToken.java 199 5/28/10 3:23
        TokenService.java 199 5/28/10 3:23 PI
      com.Pomodoro.persistent
      com.Pomodoro.persistent.test
    JRE System Library [jre6]
    Java EE 5 Libraries
    JAX-RS 1.0.2 JAXB Libraries (Project Jersey 1.0
    Referenced Libraries
    Hibernate 3.3 Annotations & Entity Manager
    Hibernate 3.3 Core Libraries
    JUnit 3
    WebRoot
```

Figure 11.5 Overview of project

As you can see in figure 11.5, in the project, we use Eclipse as IDE, SVN as version management tool, and JUnit for unit test. The project consists of 7 main components to handle different requirements. We will introduce some main components from bottom to top according to architecture introduced in last phase:
Persistent: it implements data persistent layer of architecture. Its main task is to encapsulate and maintain data objects retrieving from database.
Business: it stands for the Business logic layer of architecture. Its primary goal is to work with JSON component to deal with business logic, In other words, it is used to make original data set meaningful and representation-able.
Controller: it serves as a component used to map URLs to resources. It delivers different requests to corresponding business logic objects. Meanwhile it is responsible for entity management which restricts invalid users to visit resource that don't belong to the user.
OAuth: It is the filter in the architecture, implementing OAuth authorization protocol. Before accepting the request must be checked to show that the request is sent by a valid consumer and have been authorized by owner of resource.

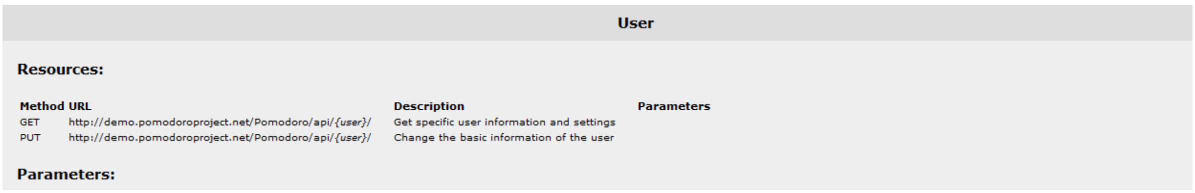
11.2.2. Figure out data set

From now on we will elaborate how we deign RESTful web services step by step.
Earlier we said that the data set would be Pomodoro technique' as we introduced before, Pomodoro project has 7 objects including: User, Project, Activity, TypeOfActivity, Pomodoro, Interruption, and Language. For some functional further uses, we added two search and setting into data set.

11.2.3. Split and publish the data.

We have generated an API file online to describe resources, their names and valid operations on them; the API file includes all the information for future development.

- User:



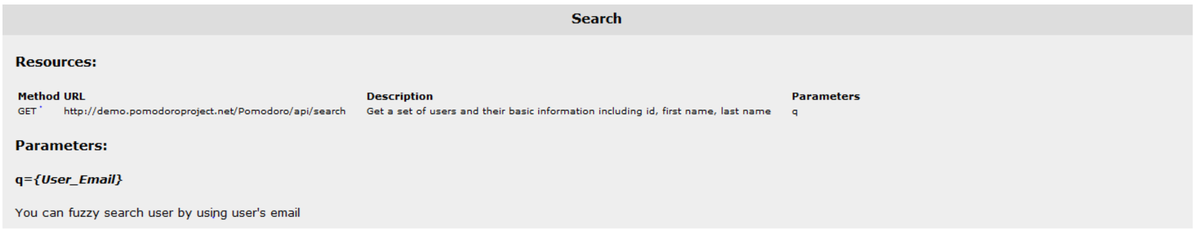
User

Resources:

Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/	Get specific user information and settings	
PUT	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/	Change the basic information of the user	

Parameters:

- Search



Search

Resources:

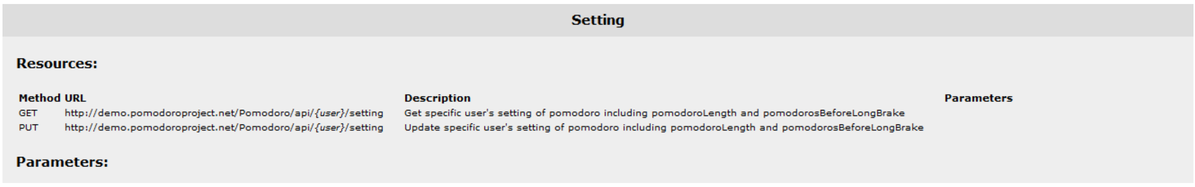
Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/api/search	Get a set of users and their basic information including id, first name, last name	q

Parameters:

q=*{User_Email}*

You can fuzzy search user by using user's email

- Setting



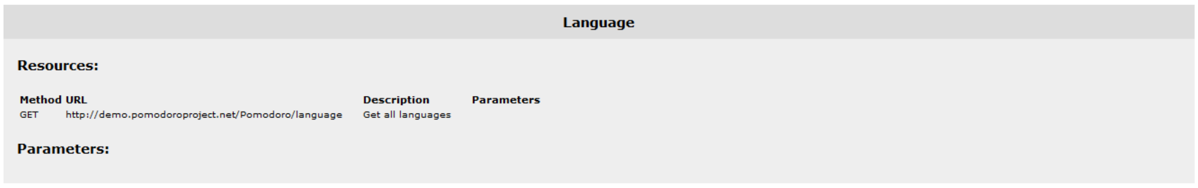
Setting

Resources:

Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/setting	Get specific user's setting of pomodoro including pomodoroLength and pomodorosBeforeLongBrake	
PUT	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/setting	Update specific user's setting of pomodoro including pomodoroLength and pomodorosBeforeLongBrake	

Parameters:

- Language



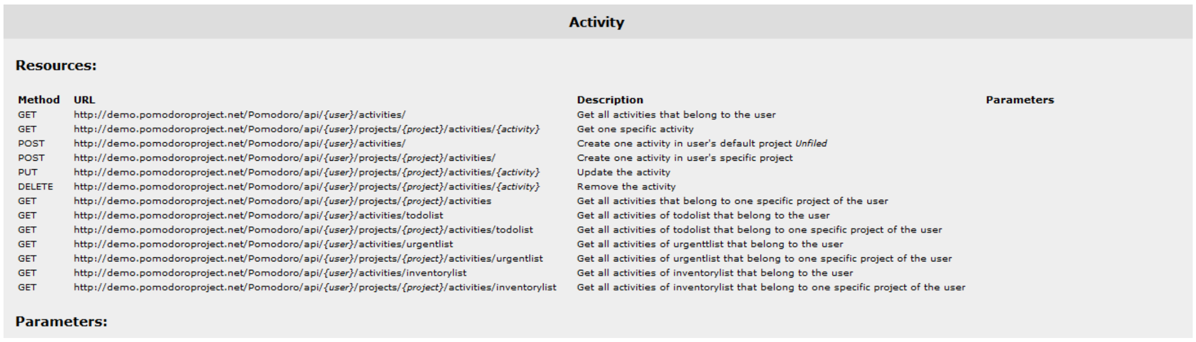
Language

Resources:

Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/language	Get all languages	

Parameters:

- Activity



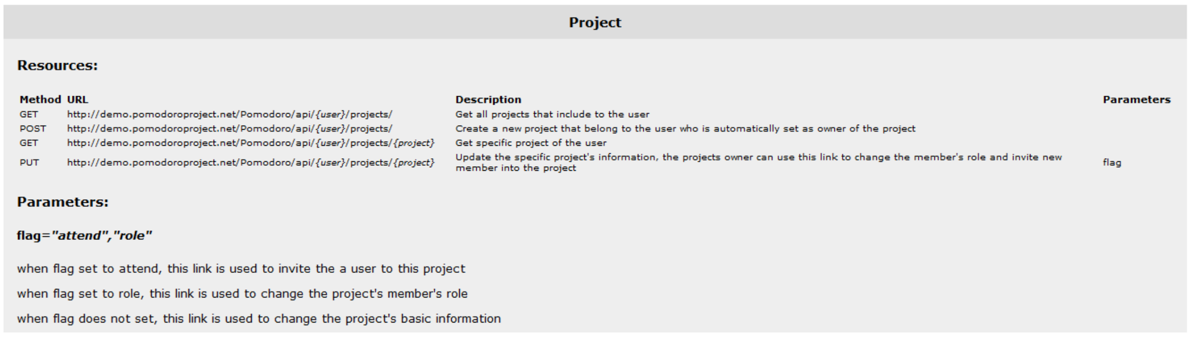
Activity

Resources:

Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/activities/	Get all activities that belong to the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*/activities/*{activity}*	Get one specific activity	
POST	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/activities/	Create one activity in user's default project *Unfiled*	
POST	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*/activities/	Create one activity in user's specific project	
PUT	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*/activities/*{activity}*	Update the activity	
DELETE	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*/activities/*{activity}*	Remove the activity	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*/activities	Get all activities that belong to one specific project of the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/activities/todolist	Get all activities of todolist that belong to the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*/activities/todolist	Get all activities of todolist that belong to one specific project of the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/activities/urgentlist	Get all activities of urgenttlist that belong to the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*/activities/urgentlist	Get all activities of urgentlist that belong to one specific project of the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/activities/inventorylist	Get all activities of inventorylist that belong to the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*/activities/inventorylist	Get all activities of inventorylist that belong to one specific project of the user	

Parameters:

- Project



Project

Resources:

Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/	Get all projects that include to the user	
POST	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/	Create a new project that belong to the user who is automatically set as owner of the project	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*	Get specific project of the user	
PUT	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/projects/*{project}*	Update the specific project's information, the projects owner can use this link to change the member's role and invite new member into the project	flag

Parameters:

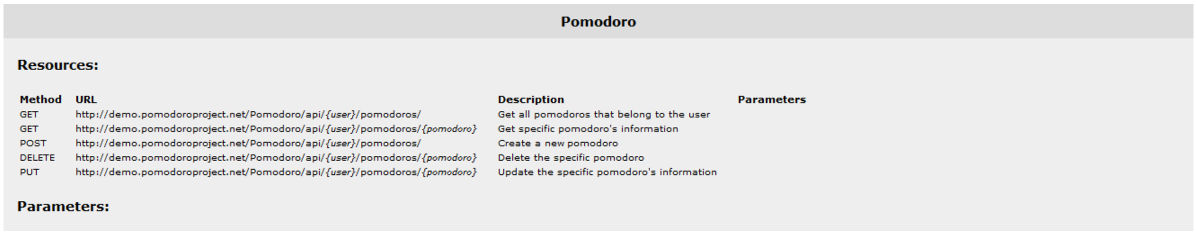
flag=*"attend","role"*

when flag set to attend, this link is used to invite the a user to this project

when flag set to role, this link is used to change the project's member's role

when flag does not set, this link is used to change the project's basic information

- Pomodoro



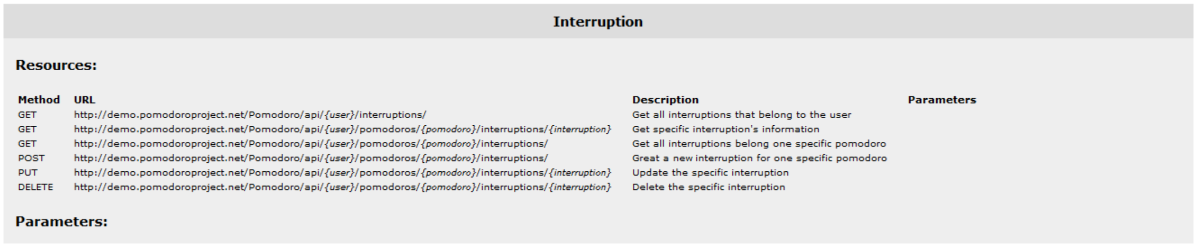
Pomodoro

Resources:

Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/	Get all pomodoros that belong to the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/*{pomodoro}*	Get specific pomodoro's information	
POST	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/	Create a new pomodoro	
DELETE	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/*{pomodoro}*	Delete the specific pomodoro	
PUT	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/*{pomodoro}*	Update the specific pomodoro's information	

Parameters:

- Interruption



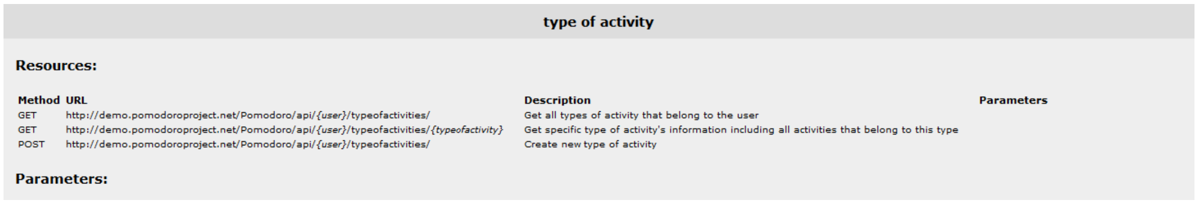
Interruption

Resources:

Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/interruptions/	Get all interruptions that belong to the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/*{pomodoro}*/interruptions/*{interruption}*	Get specific interruption's information	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/*{pomodoro}*/interruptions/	Get all interruptions belong one specific pomodoro	
POST	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/*{pomodoro}*/interruptions/	Great a new interruption for one specific pomodoro	
PUT	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/*{pomodoro}*/interruptions/*{interruption}*	Update the specific interruption	
DELETE	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/pomodoros/*{pomodoro}*/interruptions/*{interruption}*	Delete the specific interruption	

Parameters:

- TypeOfActivity



type of activity

Resources:

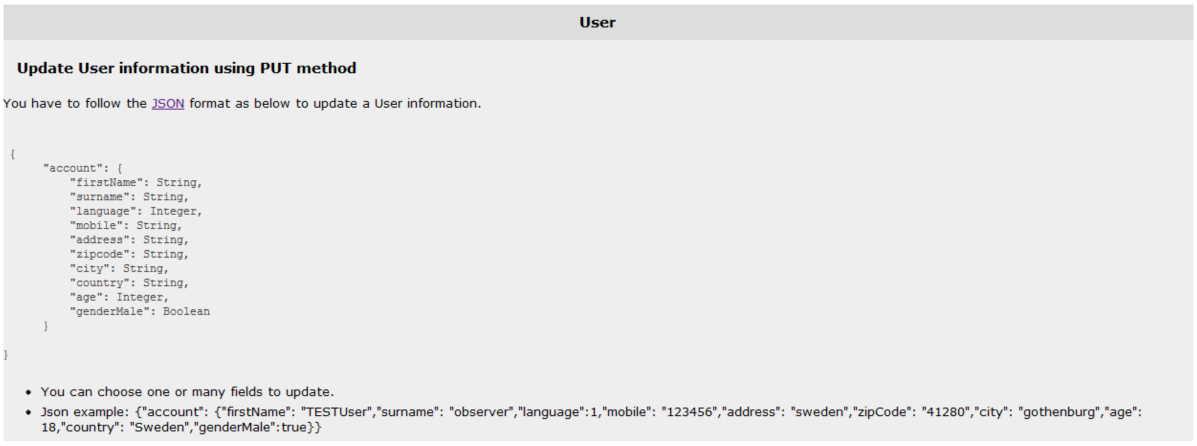
Method	URL	Description	Parameters
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/typeofactivities/	Get all types of activity that belong to the user	
GET	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/typeofactivities/*{typeofactivity}*	Get specific type of activity's information including all activities that belong to this type	
POST	http://demo.pomodoroproject.net/Pomodoro/api/*{user}*/typeofactivities/	Create new type of activity	

Parameters:

11.2.4. Design the representation

As we mentioned before, the presentations of resources in Pomodoro Project use JSON as data inter-change format. Following is the representation design of each resource:

- User



User

Update User information using PUT method

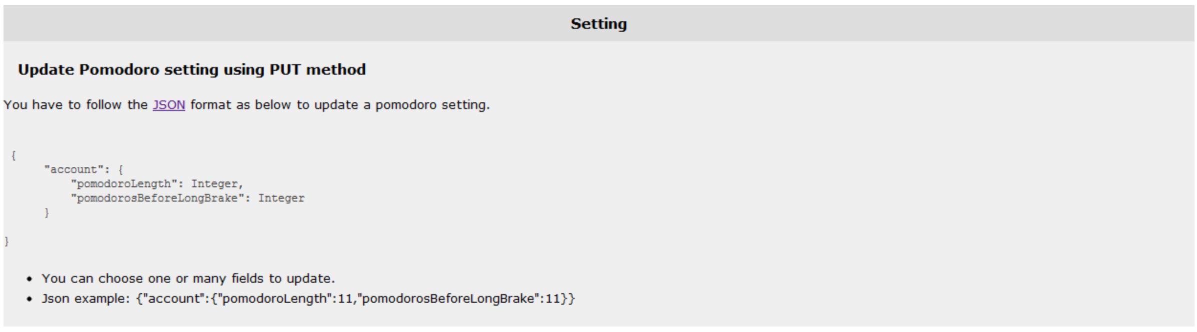
You have to follow the JSON format as below to update a User information.

```
{
     "account": {
         "firstName": String,
         "surname": String,
         "language": Integer,
         "mobile": String,
         "address": String,
         "zipcode": String,
         "city": String,
         "country": String,
         "age": Integer,
         "genderMale": Boolean
     }

}
```

- You can choose one or many fields to update.
- Json example: {"account": {"firstName": "TESTUser","surname": "observer","language":1,"mobile": "123456","address": "sweden","zipCode": "41280","city": "gothenburg","age": 18,"country": "Sweden","genderMale":true}}

- Setting



Setting

Update Pomodoro setting using PUT method

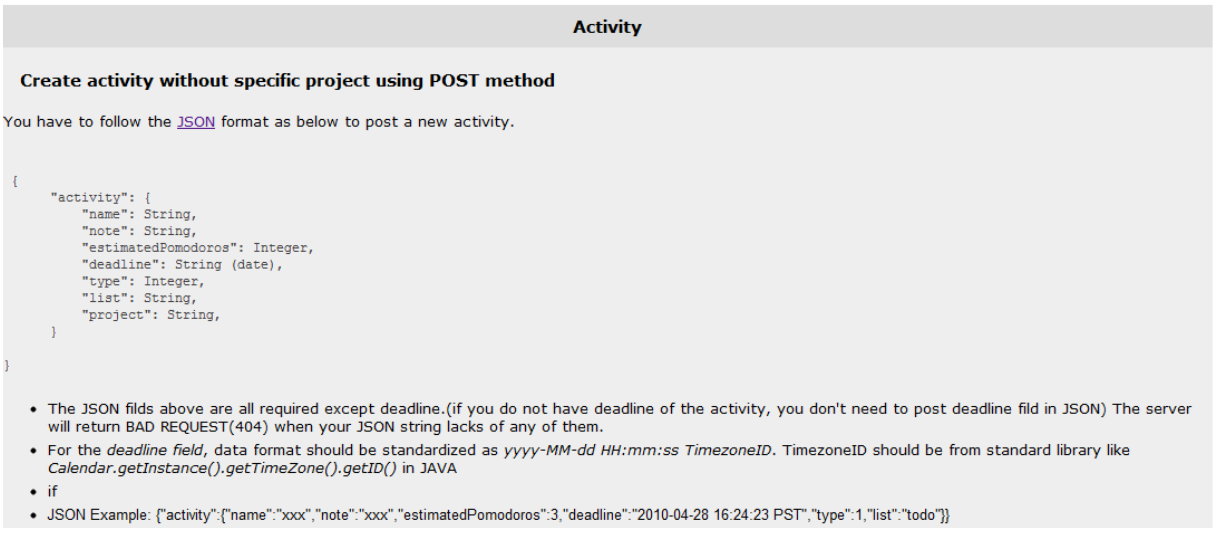
You have to follow the JSON format as below to update a pomodoro setting.

```
{
     "account": {
         "pomodoroLength": Integer,
         "pomodorosBeforeLongBrake": Integer
     }

}
```

- You can choose one or many fields to update.
- Json example: {"account":{"pomodoroLength":11,"pomodorosBeforeLongBrake":11}}

- Activity



Activity

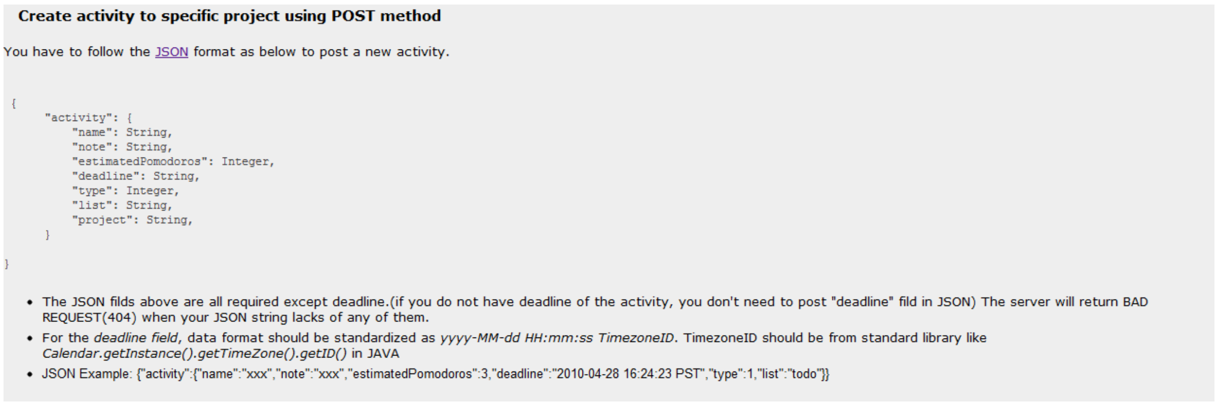
Create activity without specific project using POST method

You have to follow the JSON format as below to post a new activity.

```
{
     "activity": {
         "name": String,
         "note": String,
         "estimatedPomodoros": Integer,
         "deadline": String (date),
         "type": Integer,
         "list": String,
         "project": String,
     }

}
```

- The JSON filds above are all required except deadline.(if you do not have deadline of the activity, you don't need to post deadline fild in JSON) The server will return BAD REQUEST(404) when your JSON string lacks of any of them.
- For the *deadline field*, data format should be standardized as *yyyy-MM-dd HH:mm:ss TimezoneID*. TimezoneID should be from standard library like *Calendar.getInstance().getTimeZone().getID()* in JAVA
- if
- JSON Example: {"activity":{"name":"xxx","note":"xxx","estimatedPomodoros":3,"deadline":"2010-04-28 16:24:23 PST","type":1,"list":"todo"}}

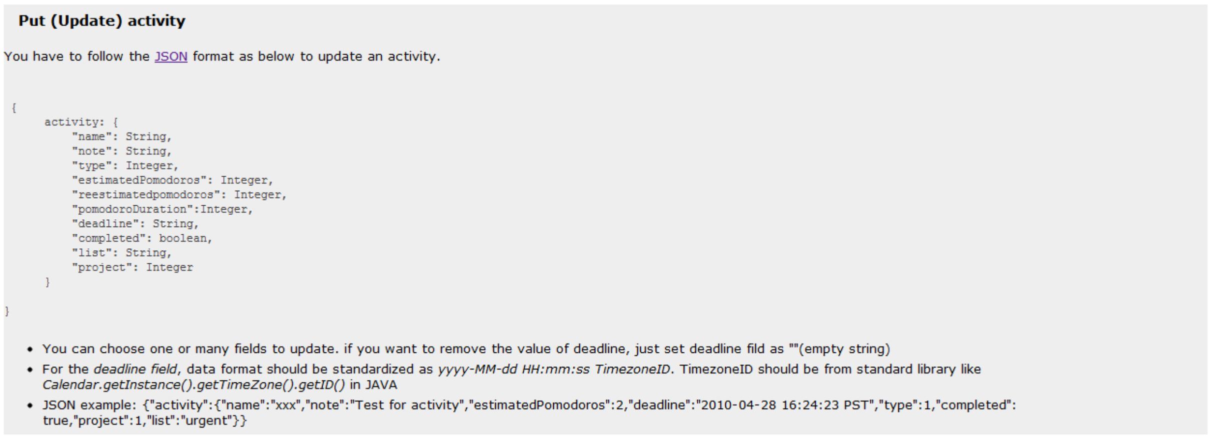


Create activity to specific project using POST method

You have to follow the JSON format as below to post a new activity.

```
{
     "activity": {
         "name": String,
         "note": String,
         "estimatedPomodoros": Integer,
         "deadline": String,
         "type": Integer,
         "list": String,
         "project": String,
     }
}
```

- The JSON filds above are all required except deadline.(if you do not have deadline of the activity, you don't need to post "deadline" fild in JSON) The server will return BAD REQUEST(404) when your JSON string lacks of any of them.
- For the *deadline field*, data format should be standardized as *yyyy-MM-dd HH:mm:ss TimezoneID*. TimezoneID should be from standard library like *Calendar.getInstance().getTimeZone().getID()* in JAVA
- JSON Example: {"activity":{"name":"xxx","note":"xxx","estimatedPomodoros":3,"deadline":"2010-04-28 16:24:23 PST","type":1,"list":"todo"}}



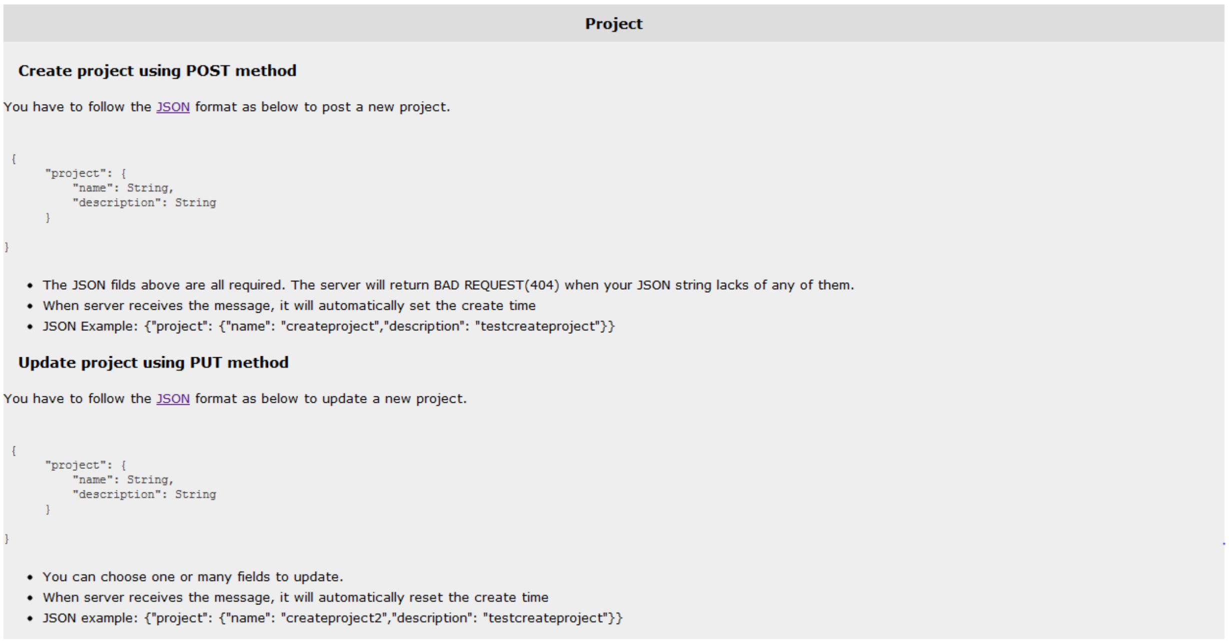
Put (Update) activity

You have to follow the JSON format as below to update an activity.

```
{
     activity: {
         "name": String,
         "note": String,
         "type": Integer,
         "estimatedPomodoros": Integer,
         "reestimatedpomodoros": Integer,
         "pomodoroDuration":Integer,
         "deadline": String,
         "completed": boolean,
         "list": String,
         "project": Integer
     }
}
```

- You can choose one or many fields to update. if you want to remove the value of deadline, just set deadline fild as ""(empty string)
- For the *deadline field*, data format should be standardized as *yyyy-MM-dd HH:mm:ss TimezoneID*. TimezoneID should be from standard library like *Calendar.getInstance().getTimeZone().getID()* in JAVA
- JSON example: {"activity":{"name":"xxx","note":"Test for activity","estimatedPomodoros":2,"deadline":"2010-04-28 16:24:23 PST","type":1,"completed": true,"project":1,"list":"urgent"}}

- Project



Project

Create project using POST method

You have to follow the JSON format as below to post a new project.

```
{
     "project": {
         "name": String,
         "description": String
     }
}
```

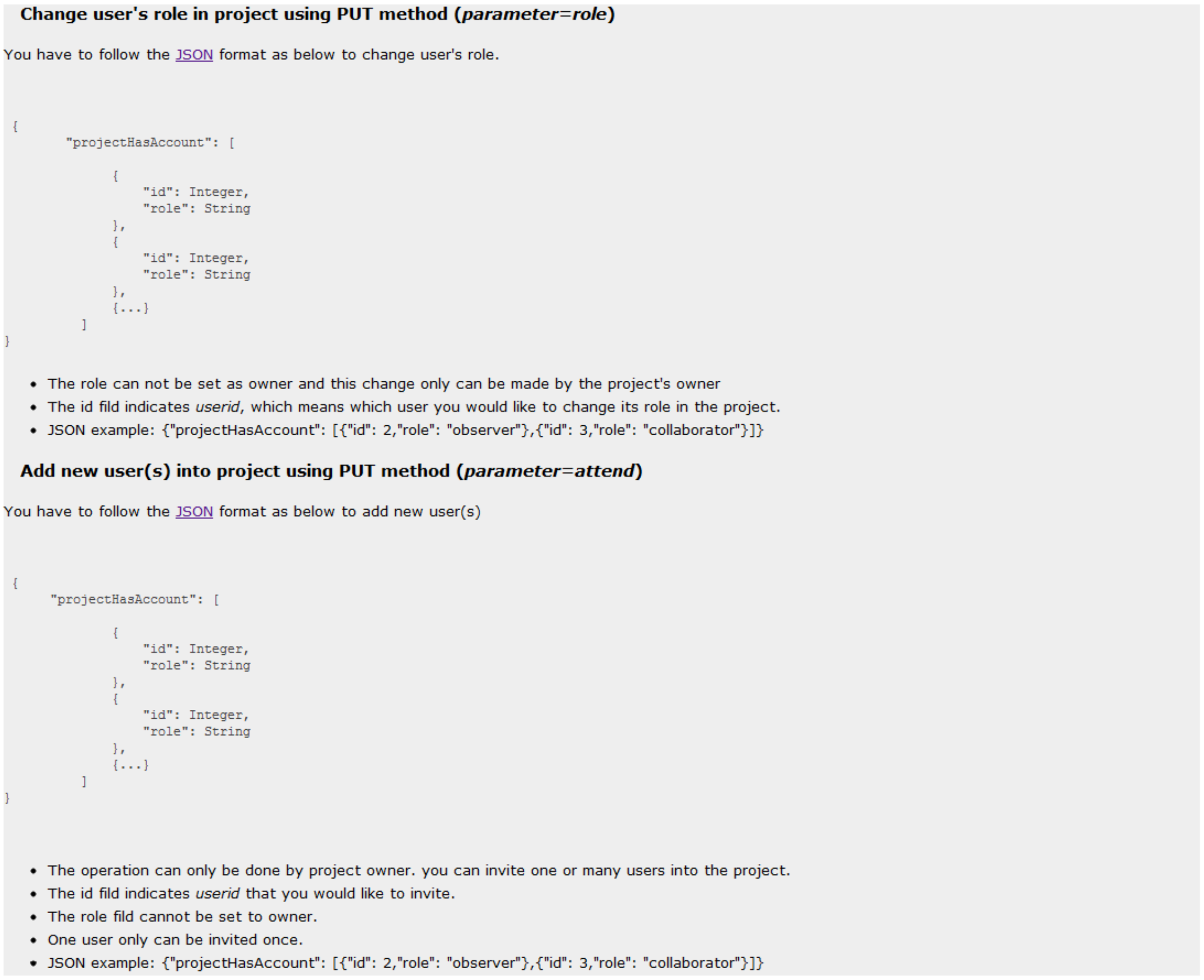
- The JSON filds above are all required. The server will return BAD REQUEST(404) when your JSON string lacks of any of them.
- When server receives the message, it will automatically set the create time
- JSON Example: {"project": {"name": "createproject","description": "testcreateproject"}}

Update project using PUT method

You have to follow the JSON format as below to update a new project.

```
{
     "project": {
         "name": String,
         "description": String
     }
}
```

- You can choose one or many fields to update.
- When server receives the message, it will automatically reset the create time
- JSON example: {"project": {"name": "createproject2","description": "testcreateproject"}}



Change user's role in project using PUT method (*parameter=role*)

You have to follow the JSON format as below to change user's role.

```
{
       "projectHasAccount": [

             {
                 "id": Integer,
                 "role": String
             },
             {
                 "id": Integer,
                 "role": String
             },
             {...}
         ]
}
```

- The role can not be set as owner and this change only can be made by the project's owner
- The id fild indicates *userid*, which means which user you would like to change its role in the project.
- JSON example: {"projectHasAccount": [{"id": 2,"role": "observer"},{"id": 3,"role": "collaborator"}]}

Add new user(s) into project using PUT method (*parameter=attend*)

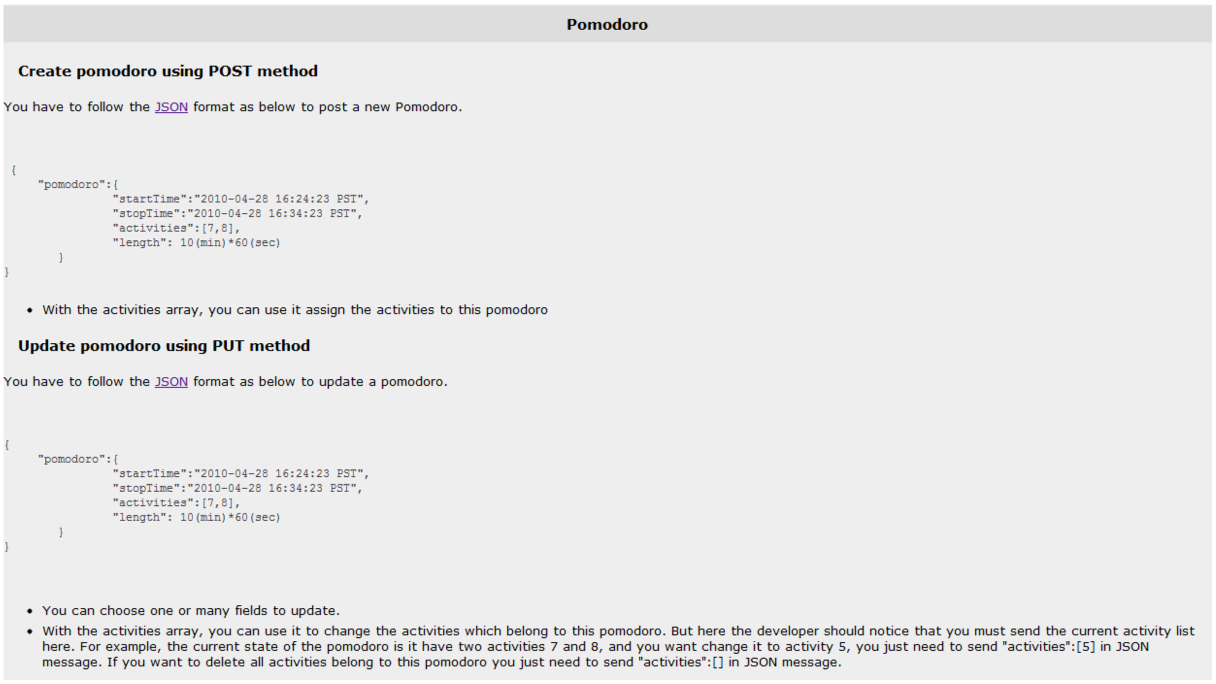
You have to follow the JSON format as below to add new user(s)

```
{
     "projectHasAccount": [

             {
                 "id": Integer,
                 "role": String
             },
             {
                 "id": Integer,
                 "role": String
             },
             {...}
         ]
}
```

- The operation can only be done by project owner. you can invite one or many users into the project.
- The id fild indicates *userid* that you would like to invite.
- The role fild cannot be set to owner.
- One user only can be invited once.
- JSON example: {"projectHasAccount": [{"id": 2,"role": "observer"},{"id": 3,"role": "collaborator"}]}

- Pomodoro



Pomodoro

Create pomodoro using POST method

You have to follow the JSON format as below to post a new Pomodoro.

```
 {
    "pomodoro":{
               "startTime":"2010-04-28 16:24:23 PST",
               "stopTime":"2010-04-28 16:34:23 PST",
               "activities":[7,8],
               "length": 10(min)*60(sec)
       }
}
```

- With the activities array, you can use it assign the activities to this pomodoro

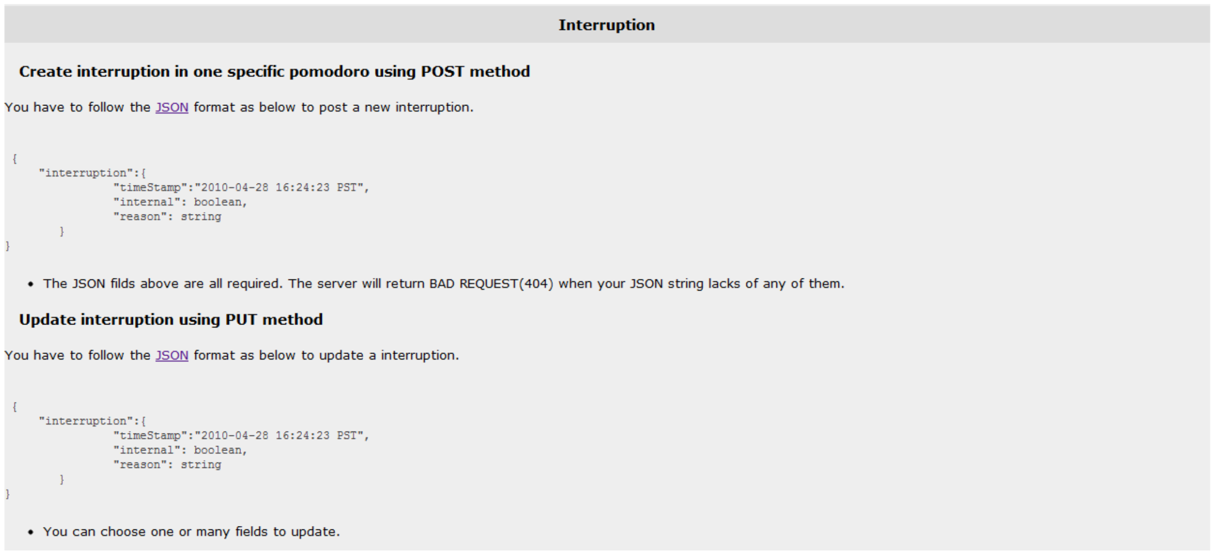
Update pomodoro using PUT method

You have to follow the JSON format as below to update a pomodoro.

```
{
    "pomodoro":{
               "startTime":"2010-04-28 16:24:23 PST",
               "stopTime":"2010-04-28 16:34:23 PST",
               "activities":[7,8],
               "length": 10(min)*60(sec)
       }
}
```

- You can choose one or many fields to update.
- With the activities array, you can use it to change the activities which belong to this pomodoro. But here the developer should notice that you must send the current activity list here. For example, the current state of the pomodoro is it have two activities 7 and 8, and you want change it to activity 5, you just need to send "activities":[5] in JSON message. If you want to delete all activities belong to this pomodoro you just need to send "activities":[] in JSON message.

- Interruption



Interruption

Create interruption in one specific pomodoro using POST method

You have to follow the JSON format as below to post a new interruption.

```
 {
    "interruption":{
               "timeStamp":"2010-04-28 16:24:23 PST",
               "internal": boolean,
               "reason": string
       }
}
```

- The JSON filds above are all required. The server will return BAD REQUEST(404) when your JSON string lacks of any of them.

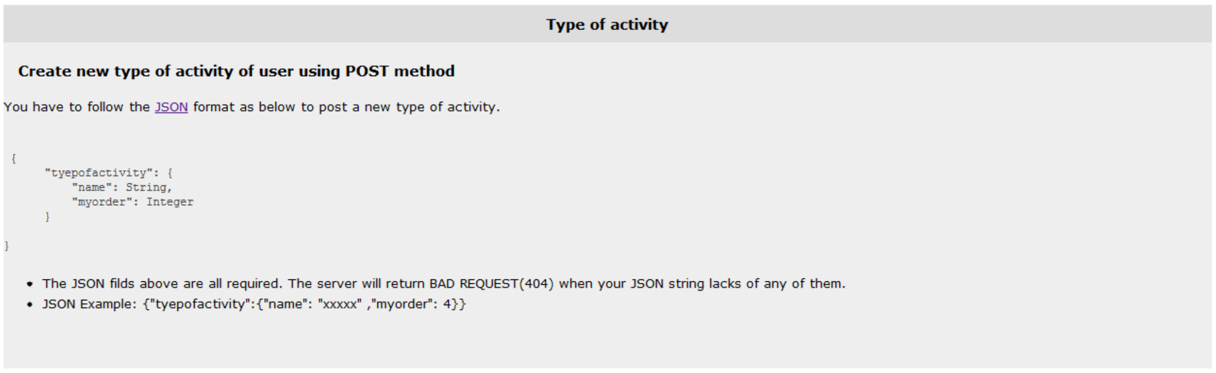
Update interruption using PUT method

You have to follow the JSON format as below to update a interruption.

```
 {
    "interruption":{
               "timeStamp":"2010-04-28 16:24:23 PST",
               "internal": boolean,
               "reason": string
       }
}
```

- You can choose one or many fields to update.

- TypeOfActivity


Type of activity

Create new type of activity of user using POST method

You have to follow the JSON format as below to post a new type of activity.

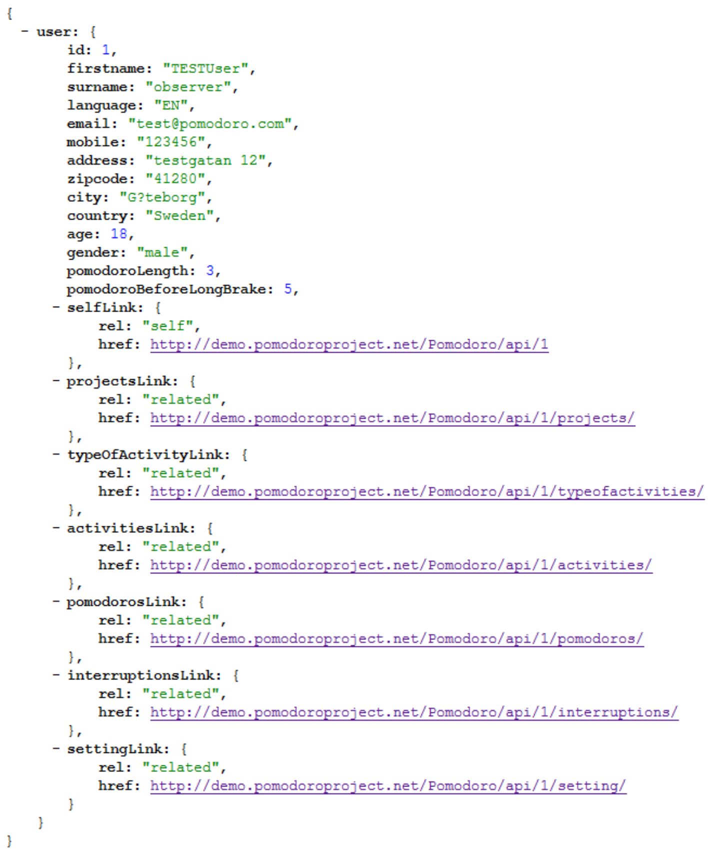
```
 {
     "tyepofactivity": {
         "name": String,
         "myorder": Integer
     }

}
```

- The JSON filds above are all required. The server will return BAD REQUEST(404) when your JSON string lacks of any of them.
- JSON Example: {"tyepofactivity":{"name": "xxxxx" ,"myorder": 4}}

11.2.5. Integrate resources using hyperlinks

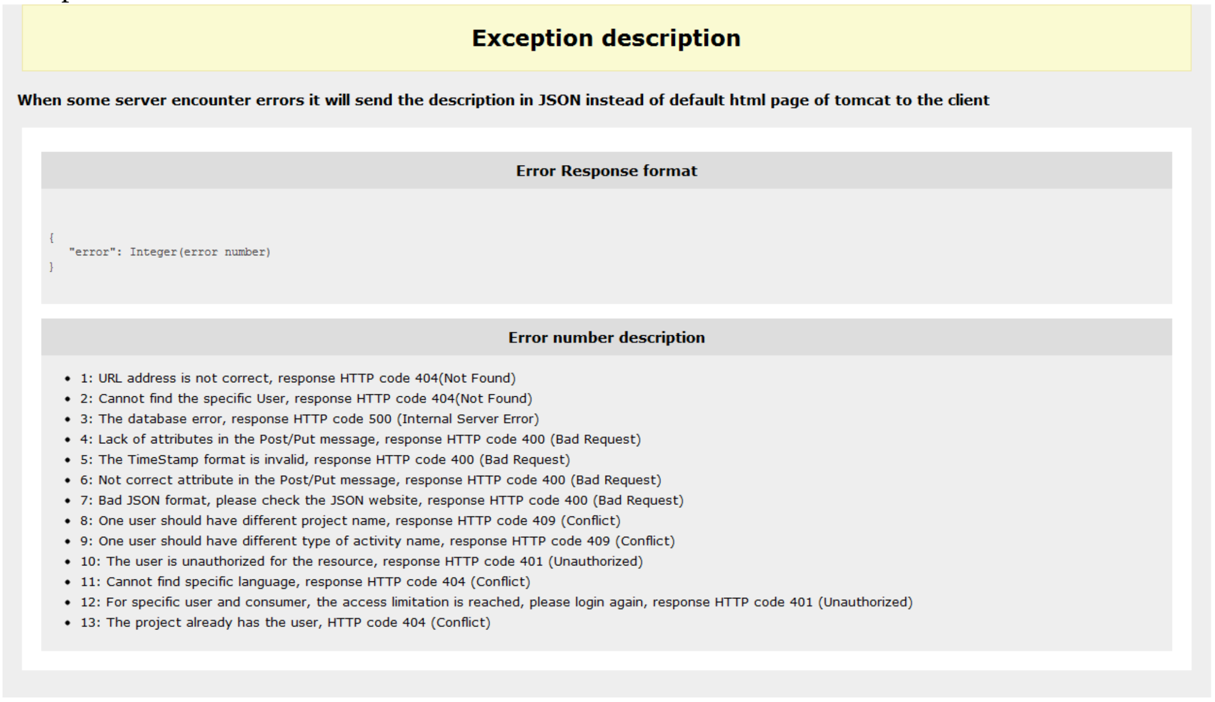
We would like to use the representation of ***User*** resource to show how Hypermedia as the Engine of Application State works.


```
{
  - user: {
        id: 1,
        firstname: "TESTUser",
        surname: "observer",
        language: "EN",
        email: "test@pomodoro.com",
        mobile: "123456",
        address: "testgatan 12",
        zipcode: "41280",
        city: "G?teborg",
        country: "Sweden",
        age: 18,
        gender: "male",
        pomodoroLength: 3,
        pomodoroBeforeLongBrake: 5,
      - selfLink: {
            rel: "self",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1
        },
      - projectsLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/projects/
        },
      - typeOfActivityLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/typeofactivities/
        },
      - activitiesLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/activities/
        },
      - pomodorosLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/pomodoros/
        },
      - interruptionsLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/interruptions/
        },
      - settingLink: {
            rel: "related",
            href: http://demo.pomodoroproject.net/Pomodoro/api/1/setting/
        }
    }
}
```

As you can see, the user representation consists of not only the basic information of the specific user but also all the URLs that are linked to several other representations which have relationship with the user. For example, we can go to see how many projects the user has, what are the activities of the user and so on.

11.2.6. Exception handling

For improving robustness and cooperation between server and client, we identified following exceptions:



Exception description

When some server encounter errors it will send the description in JSON instead of default html page of tomcat to the client

Error Response format

```
{
   "error": Integer(error number)
}
```

Error number description

- 1: URL address is not correct, response HTTP code 404(Not Found)
- 2: Cannot find the specific User, response HTTP code 404(Not Found)
- 3: The database error, response HTTP code 500 (Internal Server Error)
- 4: Lack of attributes in the Post/Put message, response HTTP code 400 (Bad Request)
- 5: The TimeStamp format is invalid, response HTTP code 400 (Bad Request)
- 6: Not correct attribute in the Post/Put message, response HTTP code 400 (Bad Request)
- 7: Bad JSON format, please check the JSON website, response HTTP code 400 (Bad Request)
- 8: One user should have different project name, response HTTP code 409 (Conflict)
- 9: One user should have different type of activity name, response HTTP code 409 (Conflict)
- 10: The user is unauthorized for the resource, response HTTP code 401 (Unauthorized)
- 11: Cannot find specific language, response HTTP code 404 (Conflict)
- 12: For specific user and consumer, the access limitation is reached, please login again, response HTTP code 401 (Unauthorized)
- 13: The project already has the user, HTTP code 404 (Conflict)

11.2.7. OAuth authorization protocol implementation

As we mentioned before, in the traditional client-server authentication model, the client uses its credentials to access its resources hosted by the server. OAuth introduces a third role to this model: the resource owner. In the OAuth model, the client (which is not the resource owner, but is acting on its behalf) requests access to resources controlled by the resource owner, but hosted by the server. In order for the client to access resources, it first has to obtain permission from the resource owner. This permission is expressed in the form of a token and matching shared-secret. The purpose of the token is to make it unnecessary for the resource owner to share its credentials with the client. Unlike the resource owner credentials, tokens can be issued with a restricted scope, limited life-time and revoked independently.
OAuth is a protocol which gives us a set of standard procedures to follow. In the Pomodoro project, we implemented the OAuth protocol as followed:
Assume we have a consumer with following information:

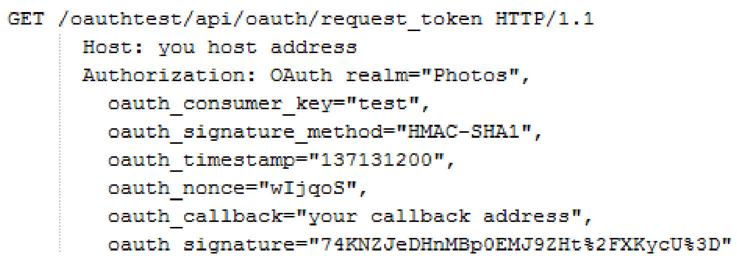
- consumer key: test
- consumer secret: e10adc3949ba59abbe56e057f20f883e

The information is generated by the server when consumers register in the server (resources owner, in our case, Pomodoro services provider).
And the user with following credentials:

- Username: test@pomodoro.com
- Password: test
- userID:1

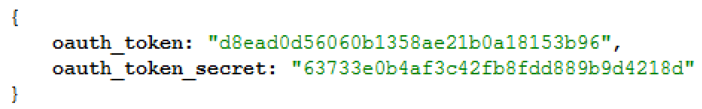
The user credentials above are forced to input by the server when users register in the server (resources owner, in our case, Pomodoro services provider).

1. When a user want to use third-part (consumer) application to access the resource on server (Pomodoro services provider), the third-part application first check whether there is a access token available, if not, the third-part application should send the HTTPS request to the server like this:



```
GET /oauthtest/api/oauth/request_token HTTP/1.1
      Host: you host address
      Authorization: OAuth realm="Photos",
        oauth_consumer_key="test",
        oauth_signature_method="HMAC-SHA1",
        oauth_timestamp="137131200",
        oauth_nonce="wIjqoS",
        oauth_callback="your callback address",
        oauth_signature="74KNZJeDHnMBp0EMJ9ZHt%2FXKycU%3D"
```

2. Then the server will verify the consumer key, the timestamp and signature, if it passed, the server will generate the response include:

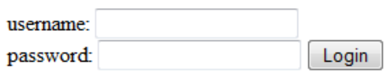


```
{
    oauth_token: "d8ead0d56060b1358ae21b0a18153b96",
    oauth_token_secret: "63733e0b4af3c42fb8fdd889b9d4218d"
}
```

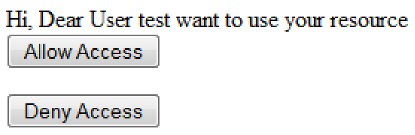
3. After the consumer received the response, the token and secret can be used to construct next step message. Then the consumer send the HTTPS request:

https://demo.pomodoroproject.net/oauthtest/api/oauth/authorize?oauth_token=d8ead0d56060b1358ae21b0a18153b96

4. The server receives the request and then checks whether this token exists, if yes, it will response 307(temporary redirect) and redirect the user to log in page.



5. The user enters the user name and password then log in. if successful; Server will redirect the user to authorization page.


Hi, Dear User test want to use your resource

6. If the user allow the consumer access his/her information, the user will click allow access button, after this the server will redirect the user to the callback address with the token and verifier, if the callback address does not include in the first step it will use the default call back address which is record when the consumer registered. For example,

http://printer.example.com/ready?null&oauth_token=d8ead0d56060b1358ae21b0a18153b96&oauth_verifier=1coxcsxmq2gmb

7. Then the consumer uses the verifier and token received above to construct the request for access token.

```
GET /oauthtest/api/oauth/access_token HTTP/1.1
Host: you host address
      Authorization: OAuth realm="Photos",
        oauth_version=1.0
        oauth_nonce=1505e1056aa27d34d9378d97d869a72e
        oauth_timestamp=1273661416
        oauth_consumer_key=test
        oauth_verifier = "1coxcsxmq2gmb"
        oauth_token=d8ead0d56060b1358ae21b0a18153b96
        oauth_signature_method=HMAC-SHA1
        oauth_signature=m%2FwdqUksTPFAZ%2FRZlVGvY7wchbA%3D
```

8. When the server receives it, it will verify all information, if passed; it will generate the access token, secret and the user's id.

```
{
   oauth_token: "2e5e3722dc1143d5a0a69f55b1a65d14"
   oauth_token_secret: "ff7acb6c29acbbc885106bdefa445942"
   userid: "1"
   url: http://demo.pomodoroproject.net/oauthtest/api/1
}
```

9. Finally, the consumer can use the access token and secret to access the user’s resources on the Pomodoro services provider.

12. Recommended architecture for RESTful web services

During our development phase, we chose implementing logic business layer by ourselves, which time-consuming. In fact, if we can use mature framework to handle the layer, that would be more productively and efficiently.

So according to our experience from the Pomodoro project, we would like to recommend following high-level architecture:

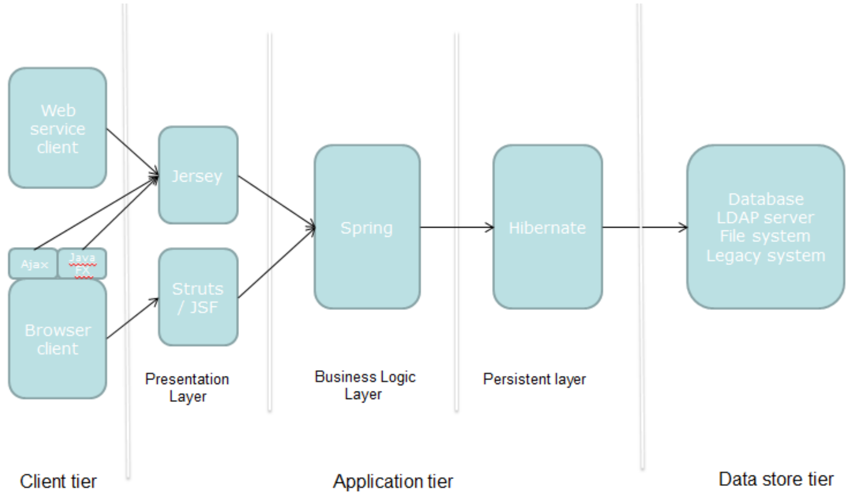


Figure 12.1 Recommended architecture for RESTful web services

Here, Spring framework helps us to handle business logic, and for our project we only offer web services, but in real world, every famous public web services has their own web site, so we suggest that there should a component (Struts/JSF) on Presentation layer to handle this. In REST world, one of hottest topic is how to integrate with legacy system. In the recommended architecture we put legacy system in data store tier; integration with legacy system task should be handled here.

13. Conclusion

This paper investigates the some issues of WS-* integration method in some circumstances. Though WS-* have been widely adopted and dominates enterprise web services integration, it lacks of interoperability and scalability. Alternatively, in the paper we introduced another integration method REpresentational State transfer, REST, which is presented as a network interaction architectural style including following primary constraints, client-server, stateless, cacheable, uniform interface, identification of resources, manipulation of resources through representations, self-descriptive messages, hypermedia as the engine of application state, and layered system. Then we used REST software architecture style to implement RESTful web services, Pomodoro services provider, intended to integrate with various platforms including iPhone, Android, web framework. Following that, we evaluated the Pomodoro services provider with those platforms. We also recommended high-level software architecture for implementing RESTful web services according to our experience. Compared to the SOAP-based integration approach, REST has many advantages such as service is addressable and can be connected to, interface is consistent, and resources can be cached. Moreover, Restful Web services is the Web services which has simple description of the document and is easy to release.

Future research in this area could possibly focus on security issues of REST and approach of resource discovery. Meanwhile, since REST has not been a standard industrial standard, so the pattern of how to design a REST application needs to be considered.

Bibliography

1. **Holly Stevens, Christy Pettey.** Gartner Says Cloud Computing Will Be As Influential As E-business. *Gartner Web site.* [Online] June 26, 2008. [Cited: 3 2, 2010.] http://www.gartner.com/it/page.jsp?id=707508.

2. *Is Cloud Computing Really Ready for Prime Time.* **Leavitt, Neal.** 1, Los Alamitos, CA, USA : IEEE Computer Society Press, 2009, Vol. 42. ISSN:0018-9162.

3. M. Armbrust, et al. (2009). `Above the Clouds: A Berkeley View of Cloud Computing'. Tech. rep.

4. Sun Microsystems. Cloud Computing. March, 2009

5. *Enterprise software as service.* **Jacobs, Dean.** 6, New York, NY, USA : ACM, 2005, Vol. 3, pp. 36-42. ISSN:1542-7730.

6. **Natis, Yefim V.** *Introducing SaaS-Enabled Application Platforms: Features, Roles and Futures.* s.l. : Gartner, 2007.

7. **Chou, Timothy.** *The End of Software: Transforming Your Business for the On Demand Future.* s.l. : Sams, 2004. ISBN:978-0672326981.

8. **Robert P. Desisto, James Holincheck, Gene Alvarez, Benoit J. Lheureux.** *Predicts 2007: Software as a Service Provides a Viable Delivery Model.* s.l. : Gartner, 2006.

9. **R. T. Fielding** (2000). *Architectural styles and the design of network-based software architectures*. Ph.D. thesis.

10. **R. T. Fielding** & **R. N. Taylor** (2002). `Principled design of the modern Web architecture'. *ACM Trans. Internet Technol.* **2**(2):115-150.

11. API Protocols, http://www.programmableweb.com/apis

12. **Peter Mell, Tim Grance.** *NIST Definition of Cloud Computing v15.* s.l. : National Institure of Standard and Technology, 2009.

13. Web service. *Wikipedia.* [Online] [Cited: 04 08, 2010.] http://en.wikipedia.org/wiki/Web_service.

14. *Roots of the REST/SOAP Debate.* **Prescod, Paul.** Montréal, Canada : 2002 Extreme Markup Languages Conference, 2002.

15. **R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter.** *Hypertext Transfer Protocol -- HTTP/1.1.* [Standards Track] s.l. : Network Working Group, 1999.

16. **Jong, Irmen de.** *Web Services/SOAP and CORBA.* 2002.

17. **David Booth, Hugo Haas, Francis McCabe.** *Web Services Architecture.* [Document] s.l. : W3C Working Group, 2004.

18. **Ian Jacobs, Norman Walsh.** *Architecture of the World Wide Web, Volume One.* [Document] s.l. : W3C Recommendation, 2004.

19. **Fiedling, et al.** *Hypertext Transfer Protocol --HTTP/1.1 Mehtod Definitions.* s.l. : Network Working Group, 1999.

20. **Leonard Richardson, Sam Ruby.** *RESTful Web Services.* s.l. : O'Reilly, 2007. ISBN:978-0-596-52926-0.

21. **Bray, Tim, et al.** *Extensible Markup Language (XML) 1.0 (Fourth Edition).* s.l. : W3C, 2006.

22. **Nurzhan Nurseitov, et al.** *Comparison of JSON and XML Data Interchange Formats: A Case Study.*

23. **Ravi Sandhu, Pierangela Samarat.** *Authentication, access control, and audit* . s.l. : ACM Computing Surveys, 1996.

24. **OAuth.** OAuth Security Advisory: 2009.1. *OAuth.* [Online] OAuth, 1 2009. [Cited: 05 03, 2010.] http://oauth.net/advisories/2009-1/.

25. **Fielding, Roy T.** Untangled musings of Roy T. Fielding. *REST APIs must be hypertext-driven.* [Online] 10 20, 2008. [Cited: 5 4, 2010.] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.